Honeywell

Experion PKS

Application Development Guide

EPDOC-XXX5-en-431A
February 2015

Release 431

Honeywell

Document	Release	Issue	Date
EPDOC-XXX5-en-431A	431	0	February 2015

Disclaimer

Contents

About this guide **11**
Server API reference **13**
Prerequisites 14
About the Server API development environment **15**
Using Microsoft Visual Studio to develop Server API applications 16
Setting up Microsoft Visual Studio for Server API applications 16
Compiling and linking C and C++ projects 18
Setting up debugging utilities and tasks 18
Disabling the default debugger (Dr Watson) 19
Folder structures for C/C++ applications 20
Multithreading considerations for Server API applications 21
Error codes in the Server API 22
Validating IEEE 754 special values 23
Implementing a Server API application **25**
Application choices 26
Programming languages 26
Type of application 26
C/C++ application template 27
Definitions 27
Initialization 27
Main body of a utility 28
Main body of a task 28
Data types 29
Writing messages to the log file 30
Server redundancy 31
Developing an OPC client 32
Developing an ODBC client 33
Controlling the execution of a Server API application **35**
Starting an application 36
Running a utility from the command line 36
Selecting an LRN for a task 36
Starting a task automatically 36
Starting a task manually 37
Activating a task 39
Activating a task on a regular basis 39
Activating a task while a point is on scan 40
Activating a task when a status point changes state 41
Activating a task when a Station function key is pressed 41
Activating a task when a Station menu item is selected 42
Activating a task when a display button is selected 42
Activating a task when a Station prompt is answered 42
Activating a task when a display is called up 42
Activating a task when a report is requested 42
Activating a task from another task 43

Testing the status of a task 44
Monitoring the activity of a task 45

Accessing server data 47

Introduction to databases 48
The server database 49
Physical structure 49
Logical structure 50
Flat logical files 50
Object-based real-time database files 53
Strings in the logical file structure 53
Ensuring database consistency 53
Accessing acquired data 55
Identifying a point 55
Identifying a parameter 55
Accessing parameter values 55
Using point lists 55
Controlling when data is acquired and processed for standard points 56
Accessing process history 58
Accessing blocks of history 58
Accessing other data 59
Accessing logical files 59
Accessing memory-resident files 59
DIRTRY (The first logical file in the server database) 60
Accessing user-defined data 61
Displaying and modifying user table data 61
Setting up user tables using the UTBBLD utility 62
UTBBLD usage notes 65
Using the database scanning software 66

Working from a Station 67

Running a task from a Station 68
Routine for generating an alarm 69
Routine for using the Station Message and Command Zones 70
Routine for printing to a Station printer 71

Developing user scan tasks 73

Designing the database for efficient scanning 75
Example user scan task 76
C/C++ version 76

Development utilities 83

ADDTSK 84
CT 85
databld 86
DBG 87
DT 88
ETR 89
FILDMP 90
FILEIO 91
REMTSK 92
TAGLOG 93
USRLRN 94

Application Library for C and C++ 95

c_almmsg_...() 99
AssignLrn() 101

c_chrint() 102
ctofstr() 103
c_dataio_...() 104
DeassignLrn() 110
c_deltsk() 111
DbletoPV() 112
dsply_lrn() 113
c_ex() 114
ftocstr() 115
c_gbload() 116
c_gdbcnt() 117
c_getapp() 118
GetGDAERRcode() 119
c_geterrno() 120
c_gethstpar_..._2() 121
c_getlrn() 124
c_getlst() 125
c_getprm() 126
c_getreq() 128
c_givlst() 130
hsc_asset_get_ancestors() 131
hsc_asset_get_children() 132
hsc_asset_get_descendents() 133
hsc_asset_get_parents() 134
hsc_em_FreePointList() 135
hsc_em_GetLastPointChangeTime() 136
hsc_em_GetRootAlarmGroups() 137
hsc_em_GetRootAssets() 138
hsc_em_GetRootEntities() 139
hsc_enumlist_destroy() 140
hsc_GUIDFromString() 141
hsc_insert_attrib() 142
    Attribute Names and Index Values 144
    Valid Attributes for a Category 146
hsc_insert_attrib_byindex() 149
hsc_IsError() 152
hsc_IsWarning() 153
hsc_lock_file() 154
hsc_lock_record() 155
hsc_notif_send() 156
hsc_param_enum_list_create() 159
hsc_param_enum_ordinal() 161
hsc_param_enum_string() 162
hsc_param_format() 163
hsc_param_limits() 165
hsc_param_subscribe() 167
hsc_param_list_create() 168
hsc_param_name() 170
hsc_param_number() 171
hsc_param_range() 172
hsc_param_type() 174
hsc_param_value() 176
hsc_param_value_of_type() 178
hsc_param_values() 179

hsc_param_value_put() 182
hsc_param_values_put() 184
hsc_param_value_save() 186
hsc_pnttyp_list_create() 188
hsc_pnttyp_name() 190
hsc_pnttyp_number() 191
hsc_point_entityname() 192
hsc_point_fullname() 193
hsc_point_get_children() 194
hsc_point_get_containment_ancestors() 195
hsc_point_get_containment_children() 196
hsc_point_get_containment_descendents() 197
hsc_point_get_containment_parents() 198
hsc_point_get_parents() 199
hsc_point_get_references() 200
hsc_point_get_referers() 201
hsc_point_guid() 202
hsc_point_name() 203
hsc_point_number() 204
hsc_point_type() 205
hsc_StringFromGUID() 206
hsc_unlock_file() 207
hsc_unlock_record() 208
HsctimeToDate() 209
HsctimeToFiletime() 210
infdouble() 211
inffloat() 212
Int2toPV() 213
Int4toPV() 214
c_intchr() 215
IsGDAerror() 216
IsGDAnoerror() 217
IsGDAwarning() 218
isinfdouble() 219
isinffloat() 220
isnandouble() 221
isnanfloat() 222
Julian/Gregorian date conversion() 223
c_logmsg() 224
c_mzero() 225
nandouble() 226
nanfloat() 227
c_oprstr_...() 228
c_pps_2() 230
c_ppsw_2() 231
c_ppv_2() 232
c_ppvw_2() 233
PritoPV() 234
c_prsend_...() 235
RealtoPV() 236
c_rqtskb...() 237
c_sps_2() 239
c_spsw_2() 240
c_spv_2() 241

c_spvw_2() 242
c_stcupd() 243
stn_num() 244
StrtoPV() 245
TimetoPV() 246
c_tmstop() 247
c_tmstrt_...() 248
c_trm04() 249
c_trmtsk() 250
c_tstskb() 251
c_upper() 252
c_wdon() 253
c_wdstrt() 254
c_wttskb() 255
Backward-compatible functions 256
c_badpar() 257
c_gethstpar_...() 258
c_pps() 261
c_ppsw() 262
c_ppv() 263
c_ppvw() 264
c_sps() 265
c_spsw() 266
c_spv() 267
c_spvw() 268
Examples 269

Network API reference 271

Prerequisites 272

Network application programming 273

About the Network API 274
    Specifying a network server in a redundant system 274
Summary of Network API functions 276
Using the Network API 277
    Determining point numbers 277
    Determining parameter numbers 278
    Accessing point parameters 278
    Accessing historical information 279
    Accessing user table data 280
    Looking up error strings 283
    Functions for accessing parameter values by name 283
Using Microsoft Visual Studio or Visual Basic to develop Network API applications 285
    Setting up Microsoft Visual Studio for Network API applications 285
    Using the Visual Basic development environment 287
    Changing packing settings when compiling C++ applications 287
Folder structures for C/C++ applications 288
Network API applications fail to run 289

Network API Function Reference 291

Functions 292
    hsc_bad_value 292
    hsc_napierrstr 293
    rgetdat 293
    rhsc_notifications 295
    rhsc_param_hist_date_bynames 297

rhsc_param_hist_offset_bynames 297
rhsc_param_hist_dates_2 301
rhsc_param_hist_offsets_2 302
rhsc_param_numbers_2 304
rhsc_param_value_bynames 306
rhsc_param_value_put_bynames 309
rhsc_param_value_put_sec_bynames 311
rhsc_param_value_puts_2 313
rhsc_param_values_2 316
rhsc_point_numbers_2 319
rputdat 321
Backward-compatibility Functions 323
hsc_napierrstr 323
rgethstpar_date 323
rgethstpar_ofst 323
rgetpnt 325
rgetval_numb 326
rgetval_ascii 326
rgetval_hist 326
rgetpntval 328
rgetpntval_ascii 328
rhsc_param_hist_dates 329
rhsc_param_hist_offsets 329
rhsc_param_value_puts 331
rhsc_param_values 334
rhsc_param_numbers 338
rhsc_point_numbers 339
rputpntval 341
rputpntval_ascii 341
rputval_hist 341
rputval_numb 342
rputval_ascii 342
Diagnostics for Network API functions 344

Batch Application Services 347
About Batch Application Services 348
Topology 348
About the Batch Application Services development environment 349
Licensing Batch Application Services 350
Security considerations 351
Automated System account 351
Console Stations 351
Event logging 351
BatchML Object Model 352
Function reference 353
IsPrimary 353
GetActivityList 353
GetActivityEntityList 356
GetActivityEntityMetadata 358
CreateActivity 361
SetPointParameterValues 363
GetPointParameterValues 365
Filtering 369
Filtering class diagrams 369
UTC offset for DateTime filtering 372

Diagnostics 373
Error codes 373
WCF exceptions 373

Batch Application Services tutorials 375

Prerequisites for a C# project 376
Creating a new C# project 376
Consuming the WCF Service 377
Adjusting app.config to support large list operations 381
Prerequisites for a C++ project 383
Creating a new C++ project 383
Adding directories and dependencies 385
Copying the Batch Application Services resources 391
Header file and namespaces 393
Simple tutorial 394
Querying the IsPrimary operation 394
Getting a list of activity entities 396
Getting more information about an activity entity 397
Creating an activity 399
Getting more information about an activity 400
Configuring an activity before starting it 405
Running the activity through its life cycle 407
Monitoring the state of the activity and reacting to state changes 409
Intermediate tutorial 412
Filtering activity lists and activity entity lists 412
Activity filter: Type 413
Activity filter: Asset 414
Activity filter: Server Base Name 414
Activity filter: Batch ID 415
Activity filter: Public Name 416
Activity filter: Tag Name 417
Activity filter: Stage 417
Activity filter: State 418
Activity filter: Create Time 419
Activity filter: Actual Start Time 420
Activity filter: Actual End Time 421
Combining multiple criteria in string filters 422
Nesting criteria for DateTimeFilter 424
Printing returned activities 426
Activity entity filter: Type 429
Activity entity filter: Asset 429
Activity entity filter: Server Base Name 430
Activity entity filter: Public Name 431
Activity entity filter: Point Build Date 431
Printing returned activity entities 432
Error scenarios and exception handling 435
Faulted proxy 435
Handling Partial Function Fail 435
Connecting to multiple servers 438
Managing redundancy 439
Installing the Batch Application Services Client Component 440
Configuring the Batch Application Services Client Component 443
Configuring your client application to redirect messages to the redundancy service 444
Using client proxies to communicate with redundant servers 445
Using Class Based Recipes 447

Batch Application Services reference **451**
BatchML schema 452
BatchInformation schema 452
MasterRecipe Type schema 453
ControlRecipe Type schema 454
BatchList schema 456
BatchListEntry schema 456
Formula Element schema 458
Parameter Element schema 458
Value Element schema 459
BatchML schema extensions 461
ReturnInformation schema 461
ElementReturn schema 461
Experion PointParam schema 461
Experion Point schema 462
BatchML Object Model class diagrams 463
Using Experion's Automation Objects **471**
Server Automation Object Model 472
HMIWeb Object Model 473
Station Scripting Objects 474
Creating an SSO 474
Registering an SSO 475
Implementing SetStation Object 475
Implementing multiple window Station support 476
Implementing SetMultiWindowStationObject 476
Implementing a Detach method 476
Station Object Model 477
Glossary **479**
Notices **489**
Documentation feedback 490
How to report a security vulnerability 491
Support 492
Training classes 493

About this guide

This guide describes how to write applications for Experion, Release 431.

Revision history

Revision	Date	Description
A	February 2015	Initial release of document.

How to use this guide

To write applications using	Go to
Server API	“Server API reference” on page 13
Network API	“Network API reference” on page 271
Batch Application Services API	“Batch Application Services” on page 347
Object Models	“Using Experion's Automation Objects” on page 471

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 273.

For:	Go to:
Prerequisites	“Prerequisites” on page 14
An introduction to the development environment	“About the Server API development environment” on page 15
An introduction to the types of applications you can develop: *tasks* and *utilities*	“Implementing a Server API application” on page 25
Information about integrating your application with Experion	“Controlling the execution of a Server API application” on page 35
A description of the Experion database, and how you access data	“Accessing server data” on page 47
Information about writing applications for Station	“Working from a Station” on page 67
Information about writing interfaces for unsupported controllers (*user scan tasks*)	“Developing user scan tasks” on page 73
Development utilities	“Development utilities” on page 83
C application library	“Application Library for C and C++” on page 95

Related topics

“Prerequisites” on page 14

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 *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.

Related topics

"Using Microsoft Visual Studio to develop Server API applications" on page 16
"Folder structures for C/C++ applications" on page 20
"Multithreading considerations for Server API applications" on page 21
"Error codes in the Server API" on page 22
"Validating IEEE 754 special values" on page 23
"Using Microsoft Visual Studio to develop Server API applications" on page 16
"Folder structures for C/C++ applications" on page 20
"Multithreading considerations for Server API applications" on page 21

Using Microsoft Visual Studio to develop Server API applications

You use to develop Server API applications.

Related topics

"About the Server API development environment" on page 15
"Setting up debugging utilities and tasks" on page 18

Setting up Microsoft Visual Studio for Server API applications

Setting up Microsoft Visual Studio involves:

- Creating a project workspace.
- Modifying the *Include Directories* and *Library Directories* for the project to include the Experion folders.
- Modifying the project settings for Experion application development.

To create a project workspace in Visual Studio

1. In the Microsoft Visual Studio application window, choose **File** > **New** > **Project**.
   The **New Project** window appears.
2. In the hierarchical list, expand **Installed**, expand **Templates**, and then click **Visual C++** .
3. At the top of the window, select the supported version of the .NET Framework.
4. Select the project type you want to develop (**Win 32 Console Application**, **MFC Application**, and so on).
5. Complete the **Name**, **Location**, and **Solution name**, and other details for the project.
6. Click **OK** to create the project.
   The **Application Wizard** appears.
7. If required, review and modify the **Application Settings**.
8. Click **Finish**.

To modify the Include Directories and Library Directories for the project

1. In the Microsoft Visual Studio application window, choose **Project** > *name* **Properties**, where *name* is the project name. If you have the *name* (the project name) item selected in the Solution Explorer, choose **Project** > **Properties**.
   The **Property Pages** window appears.
2. In the hierarchical list, expand **Configuration Properties**, and then click **VC++ Directories**.
3. Click **Include Directories**.
4. On the right-side of the **Include Directories** row, click the drop-down arrow and then click **<Edit...>**.
   The **Include Directories** dialog box appears.
5. Click the **New Line** icon, then click the browse icon and select the following folder:
   <install folder>\Honeywell\Experion PKS\server\include

   Where *<install folder>* is the location where Experion is installed.
6. Click **OK**.
7. Click **Library Directories**.
8. On the right-side of the **Library Directories** row, click the drop-down arrow and then click **<Edit...>**.
   The **Library Directories** dialog box appears.
9. Click the **New Line** icon, then click the browse icon and select the following folder:
   <install folder>\Honeywell\Experion PKS\server\lib

Where *<install folder>* is the location where Experion is installed.

10 Click **OK**.

11 In the **Property Pages** window, click **OK**.

To modify the project settings

1 In the Microsoft Visual Studio application window, choose **Project** > *name* **Properties**, where *name* is the project name. If you have the *name* (the project name) item selected in the Solution Explorer, choose **Project** > **Properties**.

The **Property Pages** window appears.

2 In the hierarchical list, expand **Configuration Properties**, expand **C/C++**, and then click **Preprocessor**.

3 Add **NT** to the **Preprocessor Definitions**.

4 In the hierarchical list, click **Code Generation**.

If this is not visible, expand **Configuration Properties** and then expand **C/C++**.

5 Click **Runtime Library**.

6 On the right-side of the **Runtime Library** row, click the drop-down arrow and then select **Multi-threaded DLL (/MD)**.

For debugging, use **Multi-threaded Debug DLL (/MDd)**. Use **Multi-threaded DLL (/MD)** only for release compiles. If you use the incorrect library for a debug compile, the error code does not propagate correctly (it will be always zero).

7 In the hierarchical list, click **General**.

If this is not visible, expand **Configuration Properties** and then expand **C/C++**.

8 Click **Additional Include Directories**.

9 On the right-side of the **Additional Include Directories** row, click the drop-down arrow and then click **<Edit...>**.

The **Additional Include Directories** dialog box appears.

10 Click the **New Line** icon, then click the browse icon and select the following folder:

<install folder>\Honeywell\Experion PKS\server\include

Where *<install folder>* is the location where Experion is installed.

11 Click **OK**.

12 In the hierarchical list, expand **Configuration Properties**, expand **Linker**, and then click **Input**.

13 Click **Additional Dependencies**.

14 On the right-side of the **Additional Dependencies** row, click the drop-down arrow and then click **<Edit...>**.

The **Additional Dependencies** dialog box appears.

15 Type `hscsrvapi.lib`, and then click **OK**.

For debugging, use *hscsrvapid.lib*. Use library *hscsrvapi.lib* only for release compiles. If you use the incorrect library for a debug compile, the error code does not propagate correctly (it will be always zero).

16 Click **Ignore All Default Libraries**.

17 On the right-side of the **Ignore All Default Libraries** row, click the drop-down arrow and then select **No**.

18 If you are developing an MFC application and want to dynamically link to MFC, complete the following steps:

a In the hierarchical list, expand **Configuration Properties**, and then click **General**.

b Click **Use of MFC**.

c On the right-side of the **Use of MFC** row, click the drop-down arrow and then select **Use MFC in a Shared DLL**.

d In the hierarchical list, expand **Configuration Properties**, expand **Linker**, and then click **Input**.

e In the **Ignore Specific Default Libraries** row, .ensure that *msvcrt* is *not* listed.

19 If you are developing an MFC application and want to statically link to MFC, complete the following steps:

a In the hierarchical list, expand **Configuration Properties**, and then click **General**.

b Click **Use of MFC**.

c On the right-side of the **Use of MFC** row, click the drop-down arrow and then select **Use MFC in a Static Library**.

d In the hierarchical list, expand **Configuration Properties**, expand **Linker**, and then click **Input**.

e Click **Ignore Specific Default Libraries**.

f On the right-side of the **Ignore Specific Default Libraries** row, click the drop-down arrow and then click **<Edit...>**.

The **Ignore Specific Default Libraries** dialog box appears.

g Type `msvcrt`, and then click **OK**.

20 Click **OK** to save your project settings.

Compiling and linking C and C++ projects

To compile and link your project in Microsoft Visual Studio, select **Build** > **Build <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 2008 SP1, 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>
```

Related topics

"Setting up debugging utilities and tasks" on page 18

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 **Debug** > **Go**).

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 .

Related topics

"Using Microsoft Visual Studio to develop Server API applications" on page 16
"Compiling and linking C and C++ projects" on page 18
"Error codes in the Server API" on page 22
"ETR" on page 89

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

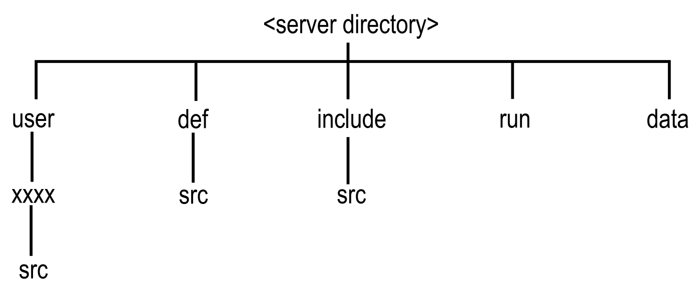
The structure shown below should be used for development of C/C++ Server API applications which will run on the server.

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 *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 *data* folder contains all server database files.



Related topics

"About the Server API development environment" on page 15

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);
```

Related topics

“About the Server API development environment” on page 15

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
    }
}
```

See also

“c_geterrno()” on page 120

Related topics

“Setting up debugging utilities and tasks” on page 18

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 211
- “inffloat()” on page 212
- “isinfdouble()” on page 219
- “isinffloat()” on page 220
- “nandouble()” on page 226
- “nanfloat()” on page 227
- “isnandouble()” on page 221
- “isnanfloat()” on page 222

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 “Figure 1: The IEEE 754 floating point binary format”), exponent is the biased exponent, and the mantissa or 'fraction' is the significand without the most significant bit.

Figure 1: 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 propogates 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 29

Implementing a Server API application

Related topics

"Application choices" on page 26
"C/C++ application template" on page 27
"Server redundancy" on page 31
"Developing an OPC client" on page 32
"Developing an ODBC client" on page 33

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.

Related topics

"Programming languages" on page 26
"Type of application" on page 26

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. (The Experion Command prompt is opened by choosing **Start** > **All programs** > **Honeywell Experion PKS** > **Server** > **Diagnostic Tools** > **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 90.
- **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.

Related topics

"Definitions" on page 27
"Initialization" on page 27
"Main body of a utility" on page 28
"Main body of a task" on page 28
"Data types" on page 29
"Writing messages to the log file" on page 30

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
defs.h	Defines system constants and some useful macros.
M4_err.h	Defines error code constants.
files	Defines all the logical file numbers of the database.
parameters	Defines all the point parameters of a point.
GBtrbtbl.h	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 *progname='myapp.c';

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

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 accesses 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(progname, '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 the command prompt window using the **printf** command and also scan messages back from the command prompt window 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 window. 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 *) &prmblk))
    {
        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
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.
lddble(int2_ptr)	Load a double value from the database.
stdble(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

Related topics

"Starting an application" on page 36
"Activating a task" on page 39
"Testing the status of a task" on page 44
"Monitoring the activity of a task" on page 45

Starting an application

These topics describe how to start an application.

Related topics

"Running a utility from the command line" on page 36
"Selecting an LRN for a task" on page 36
"Starting a task automatically" on page 36
"Starting a task manually" on page 37

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.

Related topics

"ADDTSK" on page 84
"CT" on page 85
"DBG" on page 87
"DT" on page 88
"ETR" on page 89
"REMTSK" on page 92

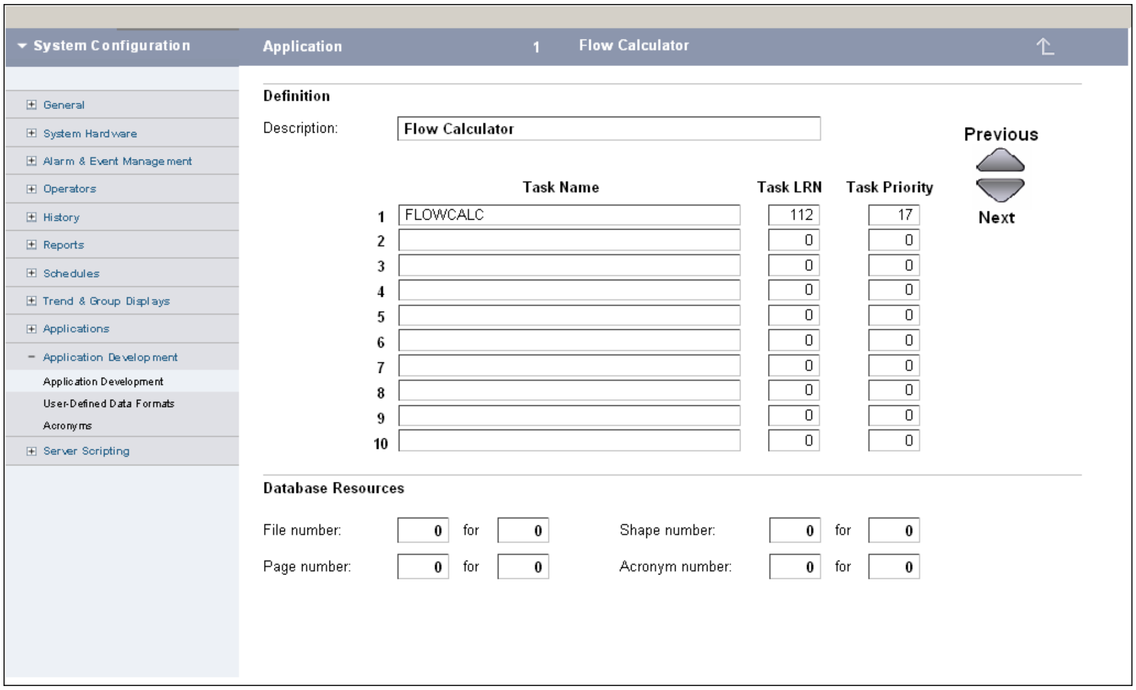
Starting a task automatically

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 **Configure** > **Application Development** > **Application 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 page 36.
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 name lrn [priority]
```

Item	Description
name	The executable file name of your task.
lrn	The LRN for the task, see “Selecting an LRN for a task” on page 36.
priority	The priority of task execution (use *0* 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 84, “ETR” on page 89 and “REMTSK” on page 92.

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 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.

Related topics

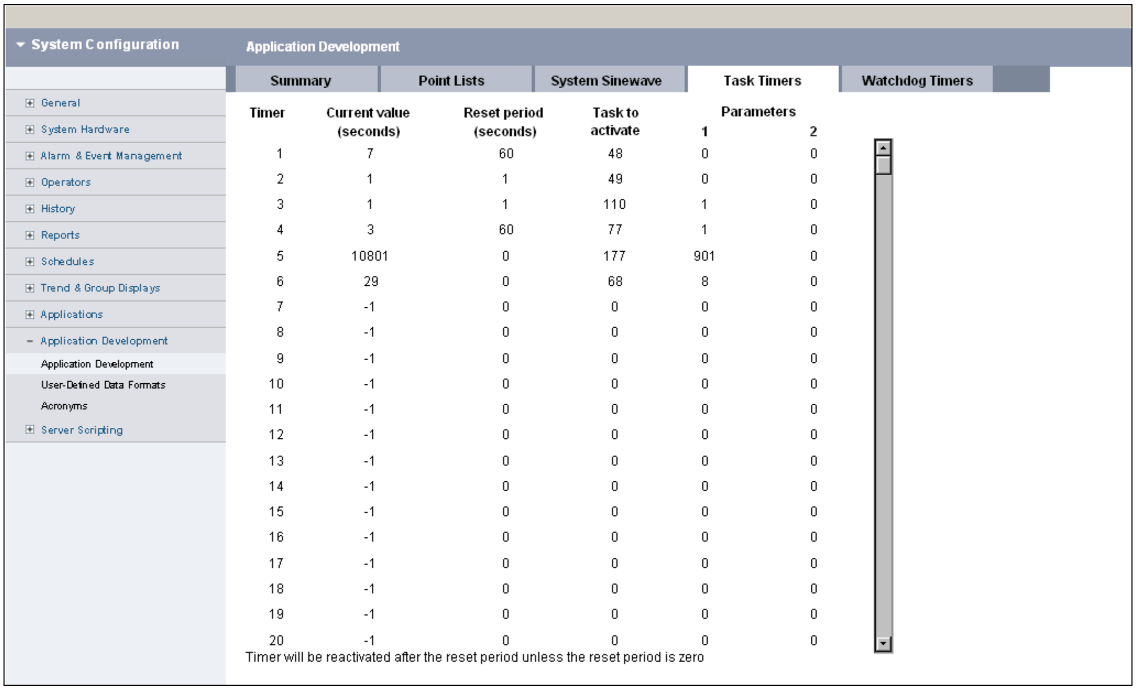
"Activating a task on a regular basis" on page 39
"Activating a task while a point is on scan" on page 40
"Activating a task when a status point changes state" on page 41
"Activating a task when a Station function key is pressed" on page 41
"Activating a task when a Station menu item is selected" on page 42
"Activating a task when a display button is selected" on page 42
"Activating a task when a Station prompt is answered" on page 42
"Activating a task when a display is called up" on page 42
"Activating a task when a report is requested" on page 42
"Activating a task from another task" on page 43
"Running a task from a Station" on page 68

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

- In Station, choose **Configure** > **Application Development** > **Task Timers** to call up the Task Timers display.



To stop the periodic requests

- 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 *Configuration Guide*.
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.

Parameter	Description
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 **alglst** utility to find a free block. See the topic titled "Algorithm blocks" in the *Server and Client Configuration Guide* 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. (See the *Application Development Guide* for details about writing custom tasks.)
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

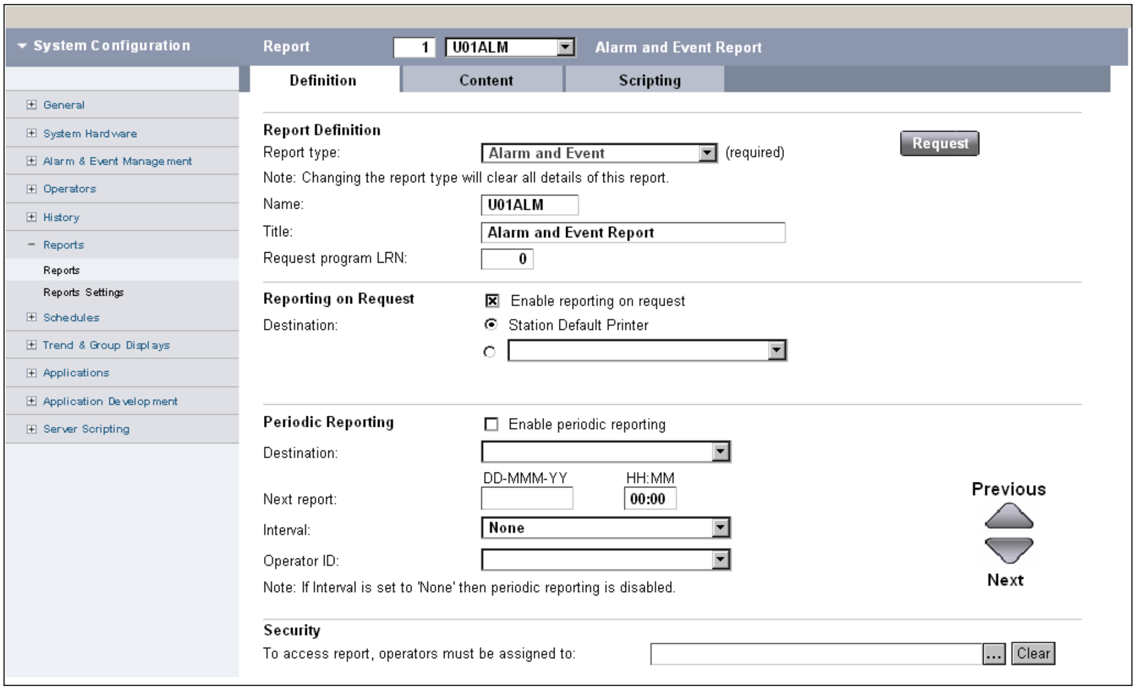
You can develop an application task that sits behind a display and performs additional processing. The display can be configured to activate your task whenever it is called up, or at regular intervals while it is visible. For details, see the topic "Defining display and shape properties (DSP)" in the *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 **Configure** > **Reports**.
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
Alarm	Generate an alarm upon failure.
Reboot	Restart the server system on failure.
Restart	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 254 and “c_wdon()” on page 253 (C and C++).

To check the watchdog timers:

- In Station, choose **Configure** > **Application Development** > **Watchdog Timers** to call up the Watchdog Timers display.

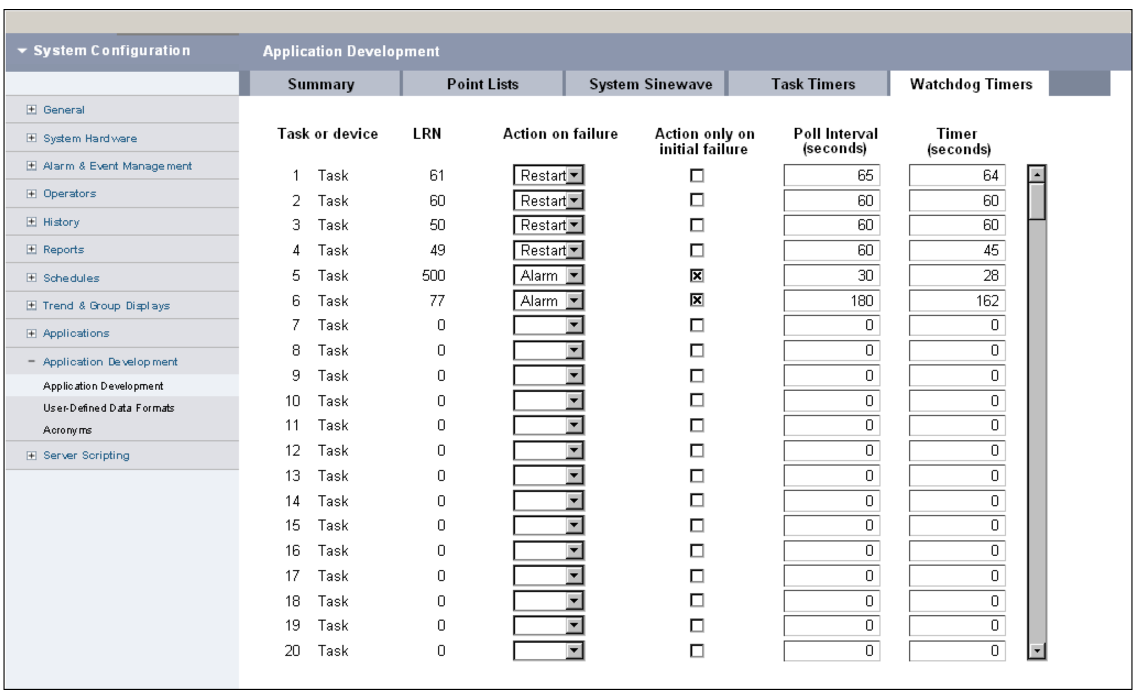


Figure 2: Watchdog timer display

Related topics

“c_wdon()” on page 253

"c_wdstrt()" on page 254

Accessing server data

Related topics

"Introduction to databases" on page 48
"The server database" on page 49
"Accessing acquired data" on page 55
"Accessing process history" on page 58
"Accessing other data" on page 59
"Accessing user-defined data" on page 61

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.
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 `<server folder>`.

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.

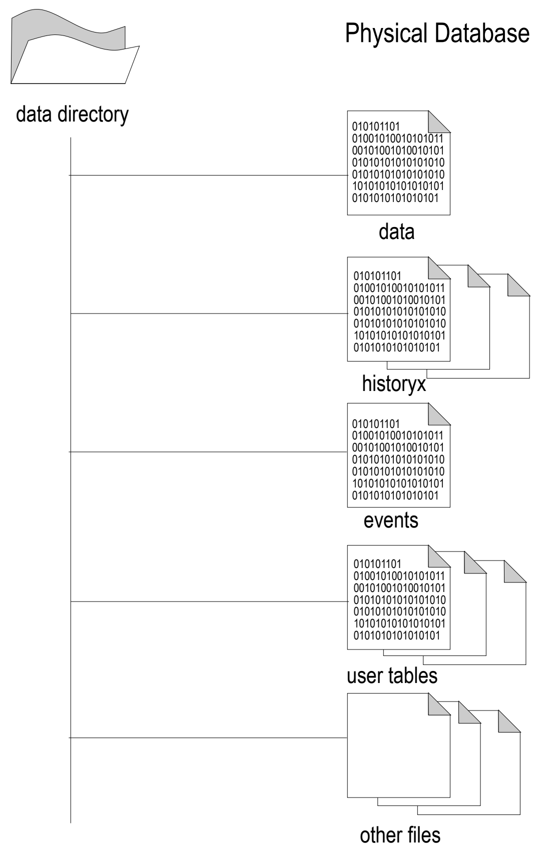


Figure 3: Physical database

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” on page 50

  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 53

  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.

Related topics

“c_dataio_...()” on page 104

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.

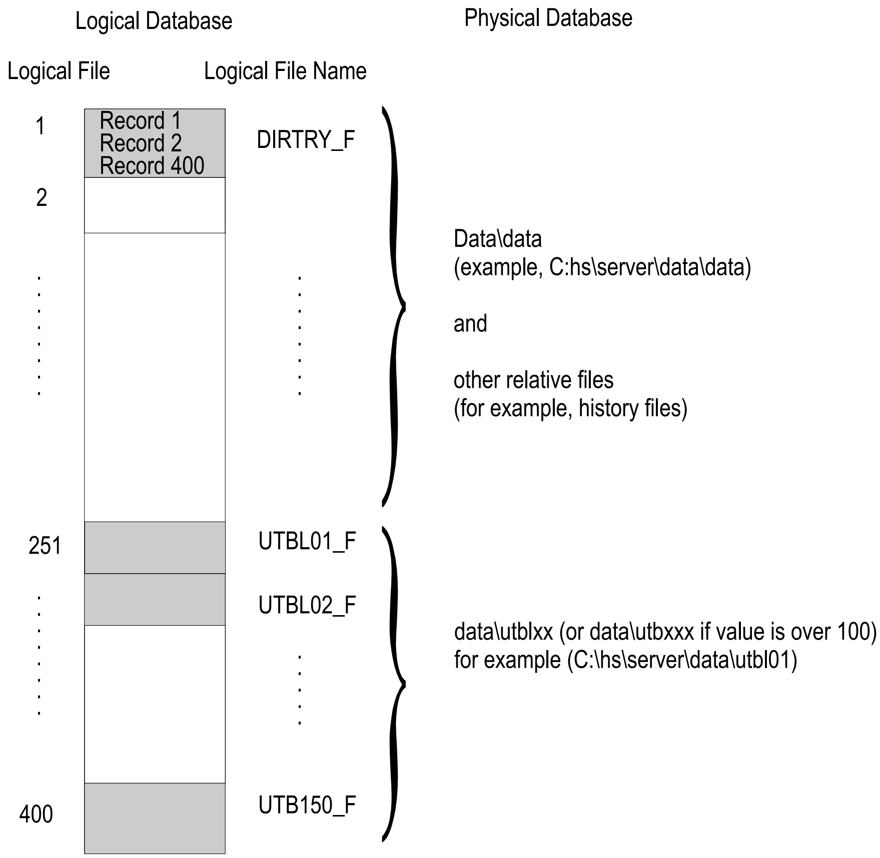


Figure 4: Logical versus physical structure of database

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.

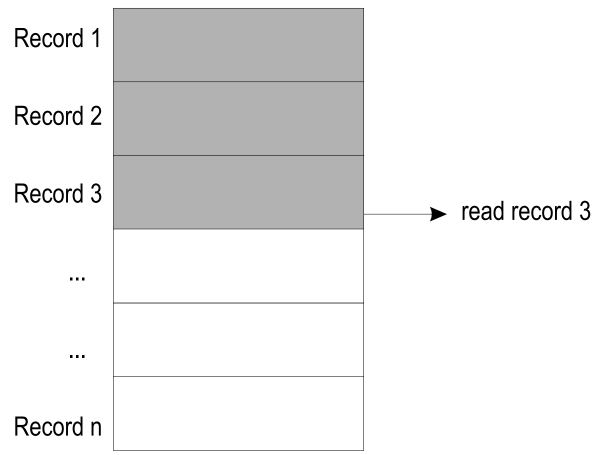


Figure 5: Relative file structure

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.

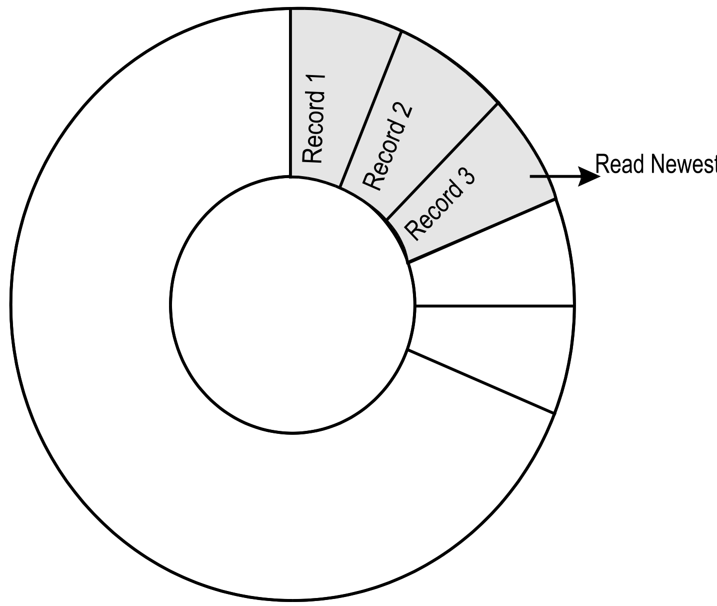


Figure 6: Circular file structure

Related topics

"Accessing user-defined data" on page 61
"c_dataio_...()" on page 104

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 page 55.

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 104, 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 102.

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 215.

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 61) 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**.

Related topics

“hsc_lock_record()” on page 155
“hsc_unlock_file()” on page 207
“hsc_unlock_record()” on page 208

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.

Related topics

"Identifying a point" on page 55
"Identifying a parameter" on page 55
"Accessing parameter values" on page 55
"Using point lists" on page 55
"Controlling when data is acquired and processed for standard points" on page 56

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 125 and “c_givlst()” on page 130 (C and C++).

To create a point list

1 Choose **Configure** > **Application Development** > **Application 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.

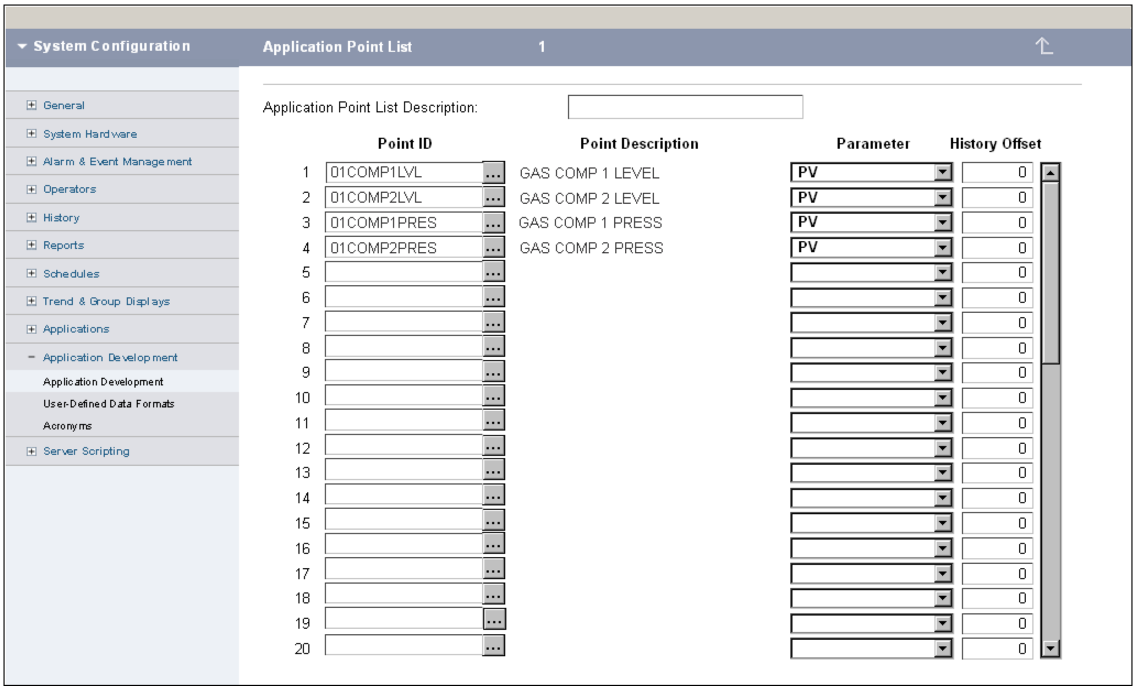


Figure 7: System configuration–Application Point display

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

Routine	Description
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*.

Related topics

"Accessing blocks of history" on page 58

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.

Related topics

"Accessing logical files" on page 59
"Accessing memory-resident files" on page 59
"DIRTRY (The first logical file in the server database)" on page 60

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 `lddbl()` and `stdbl()` 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`.

Related topics

"c_dataio_...()" on page 104

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 `GBxxxtlb[]`. 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**.

Related topics

“Displaying and modifying user table data” on page 61
“Setting up user tables using the UTBBLD utility” on page 62
“UTBBLD usage notes” on page 65
“Using the database scanning software” on page 66
“Flat logical files” on page 50
“Setting up user tables using the UTBBLD utility” on page 62

Displaying and modifying user table data

Once they've been setup, Experion gives the 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, seethe *Display Building Guide* (for DSP displays) and the *HMIWeb Display Building Guide* (for HMIWeb Displays).

User table point number storage

From Experion PKS R400, point numbers require 32 bits of data for storage; previously they required 16 bits. Therefore, when configuring new User Tables that contain point numbers, 32 bits of storage should be allocated. When upgrading from a previous version of Experion PKS, point numbers will have most likely been stored using 16 bits of storage. The User Table will need to be reconfigured to use the required 32 bits of storage. Client applications will also have to be updated to access the user table appropriately.

Setting up user tables using the UTBBLD utility

The **utbbld** 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 65.

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      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    File Number    File Type    Number of    Words per
Number                                    Records      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
```

Related topics

"Accessing user-defined data" on page 61

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. Be aware that using this option on a running server will cause all console stations connected to that server to resync.

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

Related topics

"Running a task from a Station" on page 68
"Routine for generating an alarm" on page 69
"Routine for using the Station Message and Command Zones" on page 70
"Routine for printing to a Station printer" on page 71

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."

Related topics

“Activating a task” on page 39

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.

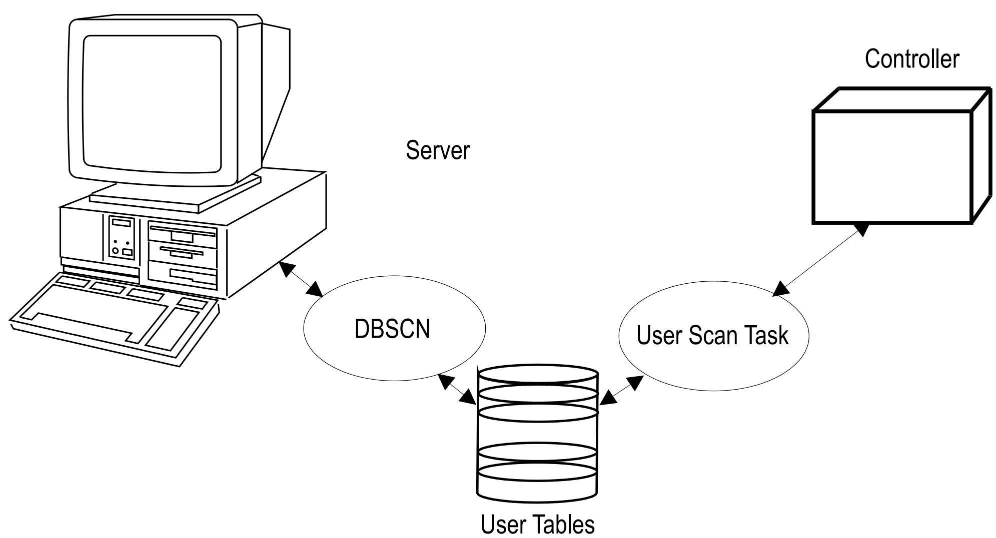


Figure 8: User scan task

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.

Related topics

"Designing the database for efficient scanning" on page 75
"Example user scan task" on page 76
"c_gdbcnt()" on page 117

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: *<server folder>/user/examples/src*.

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).

Related topics

"C/C++ version" on page 76
"Application Library for C and C++" on page 95

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 *progname="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       cntwrd;          /* 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 *)&prmblk)==0)
            {
                // GETREQ SUCCESS!

                // Test the first parameter
                switch (prmblk.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(
                                    progname,
                                    "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(
                        progname,
                        "283",
                        "DBUSER: file %d record %d write error 0x%x",
                        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",
                                prmblk.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
}
```

Related topics

"Example user scan task" on page 76
"Application Library for C and C++" on page 95

Development utilities

Tip

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

Related topics

"ADDTSK" on page 84
"CT" on page 85
"databld" on page 86
"DBG" on page 87
"DT" on page 88
"ETR" on page 89
"FILDMP" on page 90
"FILEIO" on page 91
"REMTSK" on page 92
"TAGLOG" on page 93
"USRLRN" on page 94

ADDTSK

Add application task.

Synopsis

`addtsk name lrn [priority]`

Part	Description
lrn	The LRN you have chose for the task, see 'Selecting an LRN for a task.'
priority	The priority of task execution (use 0 as a default).
name	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” on page 85 if you need to use a reserved LRN for your task.

Example

```
addtsk usrapp 111 0
```

Related topics

“Selecting an LRN for a task” on page 36

CT

Create task.

Synopsis

```
ct lrn priority -efn name
```

Part	Description
lrn	The LRN you have chose for the task, see 'Selecting an LRN for a task.'
priority	The priority of task execution (use 0 as a default).
name	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 “ADDTSK” on page 84 utility because it will check that you are not overwriting server system tasks.

Example

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

Related topics

“Selecting an LRN for a task” on page 36

databld

Description

The `databld` command is used to import and export server configuration data in XML format. `databld` is described in detail in the "Server database configuration utility (databld)" 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 *gbload()* will automatically be assigned the specified LRN.

Synopsis

dbg lrn

Part	Description
lrn	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
```

Related topics

"Selecting an LRN for a task" on page 36

DT

Delete task.

Synopsis

```
dt lrn
```

Part	Description
lrn	The LRN you have chose 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
```

Related topics

“Selecting an LRN for a task” on page 36

ETR

Enter task request

Synopsis

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

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

Description

This utility requests the specified task to be activated.

Example

```
etr 111 -arg 5
```

Related topics

"Setting up debugging utilities and tasks" on page 18
"Selecting an LRN for a task" on page 36

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 `databld` command instead of `fildmp`. `databld` 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 (databld)" 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.

Tip

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

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
lrn	The LRN you have chose 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 88 if you need to use a reserved LRN for your task.

Example

```
remtsk 111
```

Related topics

“Selecting an LRN for a task” on page 36

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 *Server and Client Configuration Guide*.

Application Library for C and C++

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

Related topics

"c_almmsg_...()" on page 99
"AssignLrn()" on page 101
"c_chrint()" on page 102
"ctofstr()" on page 103
"c_dataio_...()" on page 104
"DeassignLrn()" on page 110
"c_deltsk()" on page 111
"DbletoPV()" on page 112
"dsply_lrn()" on page 113
"c_ex()" on page 114
"ftocstr()" on page 115
"c_gbload()" on page 116
"c_gdbcnt()" on page 117
"c_getapp()" on page 118
"GetGDAERRcode()" on page 119
"c_geterrno()" on page 120
"c_gethstpar_..._2()" on page 121
"c_getlrn()" on page 124
"c_getlst()" on page 125
"c_getprm()" on page 126
"c_getreq()" on page 128
"c_givlst()" on page 130
"hsc_asset_get_ancestors()" on page 131
"hsc_asset_get_children()" on page 132
"hsc_asset_get_descendents()" on page 133
"hsc_asset_get_parents()" on page 134
"hsc_em_FreePointList()" on page 135
"hsc_em_GetLastPointChangeTime()" on page 136
"hsc_em_GetRootAlarmGroups()" on page 137
"hsc_em_GetRootAssets()" on page 138
"hsc_em_GetRootEntities()" on page 139
"hsc_enumlist_destroy()" on page 140
"hsc_GUIDFromString()" on page 141
"hsc_insert_attrib()" on page 142
"hsc_insert_attrib_byindex()" on page 149

“hsc_IsError()” on page 152
“hsc_IsWarning()” on page 153
“hsc_lock_file()” on page 154
“hsc_lock_record()” on page 155
“hsc_notif_send()” on page 156
“hsc_param_enum_list_create()” on page 159
“hsc_param_enum_ordinal()” on page 161
“hsc_param_enum_string()” on page 162
“hsc_param_format()” on page 163
“hsc_param_limits()” on page 165
“hsc_param_subscribe()” on page 167
“hsc_param_list_create()” on page 168
“hsc_param_name()” on page 170
“hsc_param_number()” on page 171
“hsc_param_range()” on page 172
“hsc_param_type()” on page 174
“hsc_param_value()” on page 176
“hsc_param_value_of_type()” on page 178
“hsc_param_values()” on page 179
“hsc_param_value_put()” on page 182
“hsc_param_values_put()” on page 184
“hsc_param_value_save()” on page 186
“hsc_pnttyp_list_create()” on page 188
“hsc_pnttyp_name()” on page 190
“hsc_pnttyp_number()” on page 191
“hsc_point_entityname()” on page 192
“hsc_point_fullname()” on page 193
“hsc_point_get_children()” on page 194
“hsc_point_get_containment_ancestors()” on page 195
“hsc_point_get_containment_children()” on page 196
“hsc_point_get_containment_descendents()” on page 197
“hsc_point_get_containment_parents()” on page 198
“hsc_point_get_parents()” on page 199
“hsc_point_get_references()” on page 200
“hsc_point_get_referers()” on page 201
“hsc_point_guid()” on page 202
“hsc_point_name()” on page 203
“hsc_point_number()” on page 204
“hsc_point_type()” on page 205
“hsc_StringFromGUID()” on page 206
“hsc_unlock_file()” on page 207
“hsc_unlock_record()” on page 208
“HsctimeToDate()” on page 209
“HsctimeToFiletime()” on page 210
“infdouble()” on page 211
“inffloat()” on page 212
“Int2toPV()” on page 213
“Int4toPV()” on page 214

"c_intchr()" on page 215
"IsGDAerror()" on page 216
"IsGDAnoerror()" on page 217
"IsGDAwarning()" on page 218
"isinfdouble()" on page 219
"isinffloat()" on page 220
"isnandouble()" on page 221
"isnanfloat()" on page 222
"Julian/Gregorian date conversion()" on page 223
"c_logmsg()" on page 224
"c_mzero()" on page 225
"nandouble()" on page 226
"nanfloat()" on page 227
"c_oprstr_...()" on page 228
"c_pps_2()" on page 230
"c_ppsw_2()" on page 231
"c_ppv_2()" on page 232
"c_ppvw_2()" on page 233
"PritoPV()" on page 234
"c_prsend_...()" on page 235
"RealtoPV()" on page 236
"c_rqtskb...()" on page 237
"c_sps_2()" on page 239
"c_spsw_2()" on page 240
"c_spv_2()" on page 241
"c_spvw_2()" on page 242
"c_stcupd()" on page 243
"stn_num()" on page 244
"StrtoPV()" on page 245
"TimetoPV()" on page 246
"c_tmstop()" on page 247
"c_tmstrt_...()" on page 248
"c_trm04()" on page 249
"c_trmtsk()" on page 250
"c_tstskb()" on page 251
"c_upper()" on page 252
"c_wdon()" on page 253
"c_wdstrt()" on page 254
"c_wttskb()" on page 255
"Backward-compatible functions" on page 256
"c_badpar()" on page 257
"c_gethstpar_...()" on page 258
"c_pps()" on page 261
"c_ppsw()" on page 262
"c_ppv()" on page 263
"c_ppvw()" on page 264
"c_sps()" on page 265
"c_spsw()" on page 266

“c_spv()” on page 267
“c_spvw()” on page 268
“Examples” on page 269
“Example user scan task” on page 76
“C/C++ version” on page 76

c_almmsg_...()

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

Note that “hsc_notif_send()” on page 156 and “hsc_insert_attrib()” on page 142 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
text	(in) pointer to a null-terminated string of text to be sent to the alarm system.
priority	(in) Message priority (see Description).
name	(in) pointer to a null-terminated string containing the alarm name (40 characters in length (alarm source)).
id	(in) pointer to a null-terminated string containing the alarm identifier (for example, PVHI, SVCHG: 20 characters in length (condition)).
level	(in) pointer to a null-terminated string containing the alarm level (for example, U, L, H, STN01).
descr	(in) pointer to a null-terminated string containing the alarm descriptor (132 characters in length).
value	(in) pointer to a null-terminated string containing the alarm value (24 characters in length).
units	(in) pointer to a null-terminated string containing the alarm units (10 characters in length).
area	(in) pointer to a null-terminated string containing the desired asset 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:

ALMMSG_LOW	Low priority
ALMMSG_HIGH	High priority
ALMMSG_URGENT	Urgent priority

c_almmsg_event_area and *c_almmsq_alarm_area* perform the same function as the *c_almmsg_event* and *c_almmsg_alarm* routines, except that the asset 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:

ALMMSG_NAME	Alarm name (equals 0)
ALMMSG_ID	Alarm ID (for example, PVHI, SVCHG)
ALMMSG_LEVEL	Alarm level (for example, L, U, H, STN01)
ALMMSG_DESCR	Alarm description
ALMMSG_VALUE	Alarm value
ALMMSG_UNITS	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 the server install folder in *user/examples/src* folder).

See also

“c_prsend_...()” on page 235

AssignLrn()

Assigns an LRN to the current thread.

C/C++ Synopsis

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

int2 AssignLrn(
int2* pLrn    // (in/out) lrn to be allocated
              // -1 == find an unused lrn
              // >0 == allocate the specified lrn
);
```

Arguments

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

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 110

"c_getlrn()" on page 124

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
chrbuf	(in) source character buffer containing ASCII
chrbuflen	(in) size of character buffer in bytes (to allow non null-terminated character buffers)
intbuf	(out) destination integer buffer
intbuflen	(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 215

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
Cstr	(in) null terminated C string
Fstr	(out) memory array for C string (*Fstr* can be the same as *Cstr*)
Flen	(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 215

"c_chrint()" on page 102

"ftocstr()" on page 115

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
filenum	(in) Server file number.

Argument	Description
filerecs	(out) Pointer to the number of records in the file. The variable referenced in the variable pointer is written to by the routine.
byreclen	(out) Pointer to the number of bytes in each record. The variable referenced in the variable pointer is written to by the routine.
cirrecnum	(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.
recnum	(in) Record number or start record number for block transfer.
numrecs	(in) Number of records to be block transferred.
location	(in) Location of data (see “Location options”).
buffer	(in/out) Pointer to the buffer variable. The variable referenced in the variable pointer is written to by the routine.
bybuflen	(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.'

c_dataio_size	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.
c_dataio_read	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.
c_dataio_write	Writes a single record from the buffer to a server file.
c_dataio_read_blk	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.
c_dataio_write_blk	Writes a number of contiguous records from the buffer to a server file.
c_dataio_queue	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.
c_dataio_dequeue	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.
c_dataio_read_newest	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.

c_dataio_read_oldest	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:

LOC_ALL	Read/write from/to memory, local disk, and backup server. Does not require any additional file handling to ensure file consistency.
LOC_MEMORY	Read/write from memory only. Requires additional file handling to ensure file consistency.
LOC_DISK	Read/write from the local disk only. Requires additional file handling to ensure file 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 120 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_recordsize = 0;

/* Determine the size of user table UTBL01 */
if (c_dataio_size(    UTBL01_F,
        &user_table1_records,
        &user_table_recordsize) == -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_recordsize) == -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 179

“hsc_param_value_put()” on page 182

“c_getlst()” on page 125

“c_givlst()” on page 130

Related topics

“Logical structure” on page 50

“Flat logical files” on page 50

“Accessing logical files” on page 59

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 120 will retrieve the error code.

See also

“AssignLrn()” on page 101

“c_getlrn()” on page 124

c_deltsk()

Marks a task for deletion.

C/C++ synopsis

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

int __stdcall c_deltsk(
    int   lrn
);
```

Arguments

Argument	Description
lrn	(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_trmtsk()” on page 250 or “c_trm04()” on page 249) 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 120 will return the following error code:

[M4_ILLEGAL_LRN]	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_deltsk(USR1LRN) == -1)
    {
    errcode = c_geterrno();
    c_logmsg(progname,'123','c_deltsk error 0x%x', errcode);
    exit(errcode);
    }
```

See also

“c_trmtsk()” on page 250

“c_trm04()” on page 249

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
pvvalue	(in) A pointer to a *PARvalue* structure.
double_val	(in) The double value to insert into the *PARvalue* 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 142 and "hsc_insert_attrib_byindex()" on page 149 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 142 will retrieve the error code.

Possible errors returned are:

BUFFER_TOO_SMALL	The pointer to the *PARvalue* is invalid, that is, null.

Example

See the examples in "hsc_insert_attrib()" on page 142 and "hsc_insert_attrib_byindex()" on page 149.

See also

"hsc_insert_attrib()" on page 142

"hsc_insert_attrib_byindex()" on page 149

"Int2toPV()" on page 213

"Int4toPV()" on page 214

"PritoPV()" on page 234

"RealtoPV()" on page 236

"StrtoPV()" on page 245

"TimetoPV()" on page 246

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

Argument	Description
pStationNumber	(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 244

c_ex()

Executes the command line.

C/C++ synopsis

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

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

Arguments

Argument	Description
command	(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 120 will retrieve the error code relevant to the command line that was executed.

ftocstr()

Converts a FORTRAN string to a C string.

C/C++ synopsis

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

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

Arguments

Argument	Description
from_str	(in) FORTRAN string to convert.
from_len	(in) length of FORTRAN string
to_str	(out) memory array for C string (*to_str* can be the same as *from_str*)
to_len	(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.

WARNING

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 215

"c_chrint()" on page 102

"ctofstr()" on page 103

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 120.

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
file	(out) file number that was controlled.
record	(out) record number that was controlled.
word	(out) word number that was controlled.
bit	(out) bit number that was controlled. 0-15 for int2 data, 0 for int4, float, dble.
width	(out) width of the data that was controlled. 1-16 for int2 data, 32 for int4 and float, 64 for dble.
value	(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.

Related topics

"Developing user scan tasks" on page 73

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

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
pGdaError	(in) pointer to the *GDAERR* structure containing the status.

Description

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

See also

"IsGDAwarning()" on page 218

"IsGDAerror()" on page 216

"IsGDAnoerror()" on page 217

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 an Experion 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_gbload() == -1)
{
    errcode = c_geterrno();
    c_logmsg(progname,'123','c_gbload error 0x%x', errcode);
    exit(errcode);
}
```

See also

"Error codes in the Server API" on page 22

c_gethstpar_..._2()

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_2
(
    int     type,
    int     date,
    float   time,
    int     numhst,
    PNTNUM  points,
    PRMNUM  params,
    int     numpnt,
    char*   archive,
    float*  values
);

int __stdcall c_gethstpar_ofst_2
(
    int     type,
    int     offset,
    int     numhst,
    PNTNUM  points,
    PRMNUM  params,
    int     numpnt,
    char*   archive,
    float*  values
);
```

Arguments

Argument	Description
type	(in) history type (see Description).
date	(in) start date of history to retrieve in Julian days (number of days since 1 Jan 1981).
time	(in) start time of history to retrieve in seconds since midnight.
offset	(in) offset from latest history value in history intervals (where offset=1 is the most recent history value).
numhst	(in) number of history values to be returned per Point.
points	(in) array of Point type/numbers to process (maximum of 100 elements).
params	(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 *def* folder (maximum 100 elements).
numpnt	(in) number of Points to be processed.
archive	(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 *<server folder>\archive*. For example, to access the files in *<server folder>\archive\ay2012m09d26h11r008*, the archive argument is *ay2012m09d26h11r008*.
values	(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 *numpnt* * *numhst*.

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.

c_gethstpar_date_2	retrieves history values from a specified date and time.
c_gethstpar_ofst_2	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 y^{th} history value for the x^{th} point.

The history type is specified by 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

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_ILLEGAL_VAL]	Illegal number of Points or history values specified.
[M4_ILLEGAL_HST]	Illegal history type or interval specified.
[M4_VAL_NOT_FND]	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 *progname = "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(progname,"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 179

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 101

“DeassignLrn()” on page 110

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
list	(in) list number (valid list numbers declared in *def/src/lstfil_def*)
values[ ]	(out) real array of values of point\parameter list. Sized *GGLNM*.
errors[ ]	(out) array of returned error codes. Sized *GGLNM*.

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 130

"c_dataio_...()" on page 104

"hsc_param_values()" on page 179

"hsc_param_value_put()" on page 182

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
paramc	(in) Parameter count. For standard use this value must be set to 3.
par1	(out) Parameter 1 value.
rqstblk	(out) Pointer to request block buffer.
rqstblk_sz	(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 120 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 237

“c_getreq()” on page 128

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
prmblk	(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 120 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 120 will return one of the following error codes:

[M4_ILLEGAL_LRN]	The calling process has not been created as a server task.
[M4_EOF_ERROR]	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 prmblk;
    if (c_gbload() == -1)
    errcode = c_geterrno();
    exit(errcode);
    while (1)
    {
        if (c_getreq((int2 *) &prmblk))
        {
            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(prmblk.param1)
            {
            case 1:
                ...
                break;
            case 2:
                ...
                break;
            } /* end switch */
        } /* if */
    } /* end while */
}
```

See also

"c_rqtskb...()" on page 237

"c_trm04()" on page 249

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
list	(in) list number (valid list numbers declared in *lstfil_def*)
values[ ]	(in) real array of values of point/parameter list. Sized *GGLNM*
errors[ ]	(out) array of returned error codes. Sized *GGLNM*

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:

[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.
[M4_PNT_ON_SCAN]	It is illegal to store the PV parameter of a point that is currently on scan.

See also

"c_getlst()" on page 125

"c_dataio_...()" on page 104

"hsc_param_values()" on page 179

"hsc_param_value_put()" on page 182

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   ASSET,
    int*     piNumAncestors,
    PNTNUM** ppAncestors
);
```

Arguments

Argument	Description
Asset	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned, and calling “c_geterrno()” on page 120 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   ASSET,
    int*     piNumChildren,
    PNTNUM** ppChildren
);
```

Arguments

Argument	Description
Asset	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned, and calling “c_geterrno()” on page 120 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   ASSET,
    int*     piNumDescendents,
    PNTNUM** ppDescendents
);
```

Arguments

Argument	Description
Asset	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned, and calling “c_geterrno()” on page 120 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   ASSET,
    int*     piNumParents,
    PNTNUM** ppParents
);
```

Arguments

Argument	Description
Asset	(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()” on page 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned, and calling “c_geterrno()” on page 120 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
pPointList	(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 120 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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned, and calling “c_geterrno()” on page 120 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**  ppRootAssets
);
```

Arguments

Argument	Description
pCount	(out) number of root assets
ppRootAssets	(out) array of root assets

Description

Returns the point numbers for all of the root assets.

The array must be cleared by calling “hsc_em_FreePointList()” on page 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned, and calling “c_geterrno()” on page 120 will retrieve the error code.

Example

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

int iNumRootAssets = 0;
PNTNUM *pRootAssets = NULL;
if (hsc_em_GetRootAssets (&iNumRootAssets, &pRootAssets) ! = 0)
    return -1;
.
.
.
hsc_em_FreePointList (pRootAssets);
```

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

Argument	Description
pCount	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned, and calling “c_geterrno()” on page 120 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 120 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) *GUID* in string format
pGUID	(out) *GUID* 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 120 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 149 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

Argument	Description
notification	(in/out) A pointer to the notification structure.
attribute_name	(in) The name of the attribute to set. See “Attribute Names and Index Values” on page 144 for a list of attribute names.

value	(in) A pointer to *PARvalue* that contains the attribute value. *PARvalue* is a union of data types and its definition is (definition from *include/src/points.h*):
	
	`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 */`
	`        } servar;`
	`    HSCTIME time;`
	`} PARvalue;`

type	(in) The value type being passed.

Description

Sets an attribute in the notification structure for use with the “hsc_notif_send()” on page 156 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 120 will retrieve one of the following error codes:

BAD_VALUE	The specified attribute value is not valid for this attribute.
BUFFER_TOO_SMALL	The pointer to the notification structure buffer is invalid, that is, null.
INV_ATTRIBUTE	The specified attribute name does not exist, or you do not have access to manipulate it.
ILLEGAL_TYPE	The specified type does not exist.
INV_AREA	The specified asset attribute is not a valid asset.
VALUE_COULD_NOT_BE_CONVERTED	The type could not be converted from the specified type to the default type for that attribute.
ATTR_NOT_IN_CAT	The specified attribute does not belong to this category. For a list of valid attributes for a category, see "Valid Attributes for a Category" on page 146.
INV_CATEGORY	The category for this notification has not been set or the passed category value is not a valid category.
CAT_ALREADY_ASSIGNED	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"
#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 149

"hsc_notif_send()" on page 156

"DbletoPV()" on page 112

"Int2toPV()" on page 213

"Int4toPV()" on page 214

"PritoPV()" on page 234

"RealtoPV()" on page 236

"StrtoPV()" on page 245

"TimetoPV()" on page 246

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
Flexible attribute Attribute name, description, and number defined in the *SysCfgSum System Attributes* display.	ALMEVTFLEXBASEIDX + <*Flexible Attribute Number*> For example, ALMEVTFLEXBASE IDX+5	DT_VAR	Flexible values. As the data type is *DT_VAR*, the optional type argument must be set.
Accessibility	ALMEVTALERTACCESSIDX	DT_INT2	
Action	ALMEVTACTIDX	DT_CHAR	The maximum size is *ALMACTUNT_SZ*.
Actor	ALMEVTACTORIDX	DT_CHAR	The actor, for example, an operator. The maximum size is *ALMEVTACTOR_SZ*.
Area code	ALMEVTACDIDX	DT_INT2 or DT_CHAR	If you specify *DT_CHAR*, you must specify the asset name. If you specify *DT_INT2*, you must specify the asset number. Must be a valid asset. If no area code attribute is created within the notification, the *hsc_notif_send* function assigns the system asset to the notification.
Asset	ALMEVTASTIDX	DT_CHAR	
Author	ALMEVTAUTHORIDX	DT_CHAR	
Block	ALMEVTBLOCKIDX	DT_CHAR	

Attribute Name	Index Value	Date Type	Description
Category	ALMEVTCATIDX	DT_INT4 or DT_CHAR	If you specify *DT_CHAR*, you must specify the category name. If you specify *DT_INT4*, you must specify the category index.
Changed Time	ALMEVTCHANGETIMEIDX	DT_TIME	
Classification	ALMEVTCLASSIDX	DT_CHAR	
Comment	ALMEVTCOMMENTIDX	DT_CHAR	The maximum size is *ALMEVTCOMMENT_SZ*.
Condition	ALMEVTCONIDX	DT_CHAR	The maximum size is *ALMEVTCON_SZ*.
Connection	ALMEVTCONNECTIDX	DT_INT2	
Criticality	ALMEVTCRITICALITYIDX	DT_CHAR	
Criticality Index	ALMEVTCRITICALITYINDEXIDX	DT_INT2	
Data Access Item	ALMEVTDAITEMIDX	DT_CHAR	
Description	ALMEVTDESIDX	DT_CHAR	A description. The maximum size is *ALMEVTDES_SZ*.
Engineering Unit	ALMEVTENGRUNITIDX	DT_CHAR	
Execution ID	ALMEVTEXEIDIDX	DT_INT8	
Field Time	ALMEVTETIMEIDX	DT_TIME	
Field Time Bias	ALMEVTFIELDBIASIDX		
Flags	ALMEVTFLAGSIDX	DT_INT2	
IOLim EE	ALMEVTIOLIMEEIDX	DT_INT2	
IsConfirmed	ALMEVTISCONFIRMEDIDX	DT_INT2	
Journal Only	ALMEVTJOURNALONLYIDX	DT_INT2	
Limit	ALMEVTLIMIDX	DT_DBLE	The alarm limit.
Link 1	ALMEVTLINK1IDX	DT_CHAR	A navigation link. The maximum size is *ALMEVTLINK_SZ*.
Link 1 Type	ALMEVTLINK1TYPEIDX	DT_INT2	Set to the default value when Link 1 is set. Link types are defined in the *almmsg.h* file.
Link 2	ALMEVTLINK2IDX	DT_CHAR	A navigation link. The maximum size is *ALMEVTLINK_SZ*.
Link 2 Type	ALMEVTLINK2TYPEIDX	DT_INT2	Set to the default value when Link 2 is set. Link types are defined in the *almmsg.h* file.
Link 3	ALMEVTLINK3IDX	DT_CHAR	A navigation link. The maximum size is *ALMEVTLINK_SZ*.
Link 3 Type	ALMEVTLINK3TYPEIDX	DT_INT2	Set to the default value when Link 3 is set. Link types are defined in the *almmsg.h* file.
Location Full Name	ALMEVTLOCFULLNAMEIDX	DT_CHAR	
Location Tag Name	ALMEVTLOCTAGNAMEIDX	DT_CHAR	
Parameter	ALMEVTPARIDX	DT_CHAR	
Prev Value	ALMEVTPREVVALIDX	DT_VAR	
Prev Value Type	ALMEVTPREVVALTYPEIDX	DT_INT2	

Attribute Name	Index Value	Date Type	Description
Priority	ALMEVTPRIIDX	DT_INT2	Includes both the priority and sub-priority value. Use the `PritoPV` function to set this attribute. Both the priority and sub-priority values must be set.
Public Name	ALMEVTPUBLICNAMEIDX	DT_CHAR	
Quality	ALMEVTQUALIDX	DT_INT2	OPC Quality value. Default value set to `c0` if not set.
Received Delay	ALMEVTRECEIVEDDELAYIDX	DT_INT4	
Reason	ALMEVTREASONIDX	DT_CHAR	The signature reason. The maximum size is `ALMEVTREASON_SZ`. *Pharma license only.*
Severity	ALMEVTSEVIDX	DT_INT4	The OPC severity.
Shelved	ALMEVTSHELVEDIDX	DT_INT2	
Shelved Reason	ALMEVTSHLVREASONIDX	DT_CHAR	
Signature Meaning 1	ALMEVTSIGNMEANIDX	DT_CHAR	The maximum size is ALMEVTSIGMEAN_SZ. *Pharma license only.*
Signature Meaning 2	ALMEVTSIGNMEAN2 IDX	DT_CHAR	The maximum size is ALMEVTSIGMEAN_SZ. *Pharma license only.*
Signature 2 Level	ALMEVTSIG2LEVELIDX	DT_CHAR	*Pharma license only.*
Source	ALMEVTSRCIDX	DT_CHAR	The point name. The maximum size is `ALMEVTSRC_SZ`.
Src Entity Name	ALMEVTSRCENTITYNAMEIDX	DT_CHAR	
Station	ALMEVTSTNIDX	DT_INT2 or DT_CHAR	If you specify `DT_INT2`, the string will be formatted. Otherwise, `DT_CHAR` is assumed. Must be a valid station.
Subcondition	ALMEVTSUBCONIDX	DT_CHAR	The maximum size is `ALMEVTCON_SZ`.
Suppressed	ALMEVTSUPPRESSEDIDX	DT_INT2	
Suppression Group	ALMEVTSUPPRESSIONGROUPID X	DT_CHAR	
Time	ALMEVTTIMEIDX	DT_TIME	
Time Bias	ALMEVTTIMEBIASIDX		
Units	ALMEVTUNTIDX	DT_CHAR	The maximum size is `ALMEVTUNT_SZ`.
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.

	Category Index[1]												
Attribute Name	**1**	**3**	**4**	**7**	**8**	**9**	**11**	**12**	**14**	**17**	**18**	**20**	**21**
Accessibility										X	X		
Action	X	X		X	X			X	X	X	X	X	X
Actor				X	X					X	X		
Area code	X	X	X	X	X	X	X	X	X	X	X	X	X
Asset	X	X	X	X	X	X	X	X	X	X	X	X	X
Author										X	X		
Block	X	X						X	X				
Category	X	X	X	X	X	X	X	X	X	X	X	X	X
Changed Time		X	X	X	X		X						
Classification										X	X		
Comment	X					X		X	X	X	X	X	X
Condition	X	X	X	X	X	X	X	X	X	X	X	X	X
Connection	X	X	X		X	X	X	X	X	X	X	X	X
Criticality	X	X						X	X				
Criticality Index	X	X						X	X				
Data Access Item	X									X	X		
Description	X	X	X	X	X	X	X	X	X	X	X	X	X
Engineering Unit	X												
Execution ID												X	X
Field Time	X	X		X	X		X	X	X	X	X	X	X
Field Time Bias	X	X		X	X		X	X	X	X	X	X	X
Flags	X	X	X	X	X	X	X	X	X	X	X	X	X
IOLim EE	X	X						X	X				
IsConfirmed							X						
Journal Only	X												
Limit	X			X	X	X	X	X	X	X	X	X	X
Link 1	X	X	X	X	X	X	X	X	X	X	X	X	X
Link 1 Type	X	X	X	X	X	X	X	X	X	X	X	X	X
Link 2	X	X	X	X	X	X	X	X	X	X	X	X	X
Link 2 Type	X	X	X	X	X	X	X	X	X	X	X	X	X
Link 3	X	X	X	X	X	X	X	X	X	X	X	X	X
Link 3 Type	X	X	X	X	X	X	X	X	X	X	X	X	X
Location Full Name	X	X	X	X	X	X	X	X	X	X	X	X	X
Location Tag Name	X	X	X	X	X	X	X	X	X	X	X	X	X
Parameter	X									X	X		
Prev Value		X		X	X	X						X	X
Prev Value Type		X		X	X	X						X	X
Priority	X	X	X				X	X	X			X	X

[1] **See the following table for the associated Category Name of each Category Index.**

	Category Index[1]												
Attribute Name	**1**	**3**	**4**	**7**	**8**	**9**	**11**	**12**	**14**	**17**	**18**	**20**	**21**
Public Name	X			X				X		X		X	
Quality	X	X	X	X	X	X	X	X	X	X	X	X	X
Received Delay	X	X						X	X				
Severity	X	X	X	X	X	X	X	X	X	X	X	X	X
Shelved	X	X		X	X					X	X		
Shelved Reason	X	X		X	X					X	X		
Signature Meaning 1			X				X						
Source	X	X	X	X	X	X	X	X	X	X	X	X	X
Src Entity Name	X	X	X	X	X	X	X	X	X	X	X	X	X
Station				X						X	X		
Subcondition	X	X	X			X	X						
Suppressed	X				X								
Suppression Group				X	X			X	X				
Time	X	X	X	X	X	X	X	X	X	X	X	X	X
Time Bias	X	X	X	X	X	X	X	X	X	X	X	X	X
Units	X	X	X	X	X	X	X	X	X	X	X	X	X
Value	X	X	X	X	X	X	X	X	X	X	X	X	X
Value Type	X	X	X	X	X	X	X	X	X	X	X	X	X

Category Index	Category Name
1	Process Alarm
3	System Alarm
4	Info Message
7	Operator Change
8	System Change
9	SOE
11	Confirmable Message
12	Process Event
14	System Event
17	Process Alert
18	Simple Alert
20	Batch Event
21	Procedure Event

[1] **See the following table for the associated Category Name of each Category Index.**

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 142 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
notification	(in/out) A pointer to the notification structure.
attribute_index	(in) The index value of the attribute to insert. See “Attribute Names and Index Values” on page 144 for a list of index values.

value	(in) A pointer to *PARvalue* that contains the attribute value. *PARvalue* is a union of data types and its definition is (definition from *include/src/points.h*):
	
	`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 */`
	`        } servar;`
	`    HSCTIME time;`
	`} PARvalue;`

type	(in) The value type being passed.

Description

Sets an attribute in the notification structure for use with the “hsc_notif_send()” on page 156 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 120 will retrieve the error code.

The possible errors returned are:

BAD_VALUE	The specified attribute value is not valid for this attribute.
BUFFER_TOO_SMALL	The pointer to the notification structure buffer is invalid, that is, null.
INV_ATTRIBUTE	The specified attribute name does not exist, or you do not have access to manipulate it.
ILLEGAL_TYPE	The specified type does not exist.
INV_AREA	The specified asset attribute is not a valid asset.
VALUE_COULD_NOT_BE_CONVERTED	The type could not be converted from the specified type to the default type for that attribute.
ATTR_NOT_IN_CAT	The specified attribute does not belong to this category. For a list of valid attributes for a category, see “Valid Attributes for a Category” on page 146.
INV_CATEGORY	The category for this notification has not been set or the passed category value is not a valid category.
CAT_ALREADY_ASSIGNED	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 112

"hsc_insert_attrib()" on page 142

"hsc_notif_send()" on page 156

"Int2toPV()" on page 213

"Int4toPV()" on page 214

"PritoPV()" on page 234

"RealtoPV()" on page 236

"StrtoPV()" on page 245

"TimetoPV()" on page 246

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
code	(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 120 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 216 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()” on page 153

“IsGDAerror()” on page 216

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	(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 120 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 216 to check this value for an error.

Diagnostics

This routine returns *TRUE* (*-1*) *code* indicates an warning condition, otherwise it returns *FALSE* (*0*).

See also

“hsc_IsError()” on page 152

“IsGDAwarning()” on page 218

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
file	(in) server file number.
delay	(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 53.

Diagnostics

Upon successful completion a value of *0* is returned. Otherwise, *-1* is returned and calling “c_geterrno()” on page 120 will retrieve one of the following error codes:

[FILLCK]	File locked to another task
[RECLCK]	Record locked to another task
[DIRLCK]	File's directory locked to another task
[BADFIL]	Illegal file number specified

See also

“hsc_lock_record()” on page 155

“hsc_unlock_file()” on page 207

“hsc_unlock_record()” on page 208

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
file	(in) server file number.
record	(in) record number (see description).
delay	(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 120 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 154

"hsc_unlock_file()" on page 207

"hsc_unlock_record()" on page 208

Related topics

"Ensuring database consistency" on page 53

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
notification	(in) A pointer to the notification structure.
mode	(in) The mode to send the notification. • *RAISE* sends the notification in the unacknowledged and off-normal state. • *RAISE_NORMALIZED* sends the notification in the unacknowledged and normal state. • *NORMALIZE* changes the state of a previous notification with identical source and condition, from off-normal to normal.

Description

Sends a notification (as created using the “hsc_insert_attrib()” on page 142 and “hsc_insert_attrib_byindex()” on page 149 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 120 will retrieve one of the following errors:

BUFFER_TOO_SMALL	The pointer to the notification is invalid, that is, null.
INV_CATEGORY	The notification does not have a valid category set.
M4_QEMPTY	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 ALMMSG_HIGH and sub-priority 0
if (hsc_insert_attrib(&myNotification, "Priority", PritoPV(ALMMSG_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 142

"hsc_insert_attrib_byindex()" on page 149

"DbletoPV()" on page 112
"Int2toPV()" on page 213
"Int4toPV()" on page 214
"PritoPV()" on page 234
"RealtoPV()" on page 236
"StrtoPV()" on page 245
"TimetoPV()" on page 246

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
point	(in) point number
param	(in) point parameter number

list	(in/out) pointer to an enumeration list of parameters *enumlist* is defined as (definition from *include/src/dictionary.h*):
	
	`typedef struct`
	`{`
	`    int    value;`
	`    char*  text;`
	`} enumlist;`
	
	• *value* is the ordinal value of the enumeration
	• *text* is the null terminated string containing the enumeration text

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 page 159 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 140 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()” on page 161

“hsc_enumlist_destroy()” on page 140

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
point	(in) point number
param	(in) point parameter number
string	(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 120.

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

Argument	Description
point	(in) point number
param	(in) point parameter number
ordinal	(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 120 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
point	(in) point number
param	(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 174

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
point	(in) point number
param	(in) parameter number
min	(out) minimum value
max	(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, &limit Max) != 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 174

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
number	(in) number of entries in lists
points	(in) list of point numbers
params	(in) list of parameter numbers
period	(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 179

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
point	(in) point number, specify 0 for all parameters of all point types

list	(in/out) pointer to an enumeration list of parameters *enumlist* is defined as (definition from *include/src/dictionary.h*):
	
	`typedef struct`
	`{`
	`    int    value;`
	`    char*  text;`
	`} enumlist;`
	
	• *value* 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 *hsc_param_number*.
	• *text* is the null terminated string containing the parameter name

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 page 168 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 140 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 %20s is %5d',
list[i].text,
list[i].value);
```

See also

“hsc_enumlist_destroy()” on page 140

“hsc_param_number()” on page 171

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

Argument	Description
point	(in) point number
param	(in) parameter number
name	(out) parameter name
namelen	(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()” on page 171

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
point	(in) point number
name	(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 can not be found or an error occurs *0* will be returned, and calling “c_geterrno()” on page 120 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 170

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
point	(in) point number
param	(in) parameter number
min	(out) minimum value
max	(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 165

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
point	(in) point number
param	(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 163

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

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>point</td><td>(in) point number</td></tr>
<tr><td>param</td><td>(in) parameter number</td></tr>
<tr><td>offset</td><td>(in) point parameter offset (for history parameters).</td></tr>
<tr><td>value</td><td>(out) value union

PARvalue is a union of data types and is defined as follows (definition from include/src/points.h):<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></td></tr>
<tr><td>type</td><td>(out) value data type (defined in the parameters file)</td></tr>
</table>

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 179 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_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                   */
```

See also

"hsc_param_values()" on page 179

"hsc_param_number()" on page 171

"hsc_point_number()" on page 204

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
point	(in) point number
param	(in) parameter number
offset	(in) point parameter offset (for history parameters).

value	(out) value union
	PARvalue is a union of data types and is defined as follows (definition from include/src/points.h):
	`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;`

type	(out) value data type (defined in the *parameters* 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()" on page 179

"hsc_param_number()" on page 171

"hsc_point_number()" on page 204

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

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>count</td><td>(in) the number of point parameters in the list to get.</td></tr>
<tr><td>period</td><td>(in) subscription period in milliseconds for the point parameters. Use the constant HSC_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 HSC_READ_DEVICE if you want to force Experion to poll the controller again. The subscription period will not be applied to standard point types.</td></tr>
<tr><td>points</td><td>(in) the list of point numbers.</td></tr>
<tr><td>params</td><td>(in) the list of point parameter numbers.</td></tr>
<tr><td>offsets</td><td>(in) the list of point parameter offsets (for history parameters).</td></tr>
<tr><td>values</td><td>(out) the list of values. PARvalue is a union of all possible value data types and is defined as follows (definition from include/src/points.h):
<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></td></tr>
<tr><td>types</td><td>the list of value types.</td></tr>
<tr><td>statuses</td><td>(out) a list containing the status of the get for each point parameter.</td></tr>
</table>

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 176
“hsc_param_number()” on page 171
“hsc_point_number()” on page 204

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
point	(in) point number
param	(in) point parameter number
offset	(in) point parameter offset (for history parameters)

value	(in) value. *PARvalue* is a union of data types and defined as (definition from *include/src/points.h*):
	`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;`

type	(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 171

“hsc_point_number()” on page 204

“hsc_param_values_put()” on page 184

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
count	(in) the number of point parameters in the list to control
points	(in) the list of point numbers
params	(in) the list of point parameter numbers
offsets	(in) the list of point parameter offsets (for history parameters)

values	(in) the list of values. *PARvalue* is a union of data types and defined as (definition from *include/src/points.h*):
	
	`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;`

types	(in) the list of value types.
statuses	(out) a list containing the status of the put for each point parameter. GDAERR is defined in <hsctypes.h>.
security	(in) GDASECURITY is defined in <hsctypes.h>. 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 171
"hsc_point_number()" on page 204
"hsc_param_value_put()" on page 182

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
point	(in) point number
param	(in) point parameter number
offset	(in) point parameter offset (for history parameters)

value	(in) value *PARvalue* is a union of data types and is defined as (definition from *include/src/points.h*):
	`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;`

type	(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 120 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 182

"hsc_param_values_put()" on page 184

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

list	(in/out) pointer to an enumeration array for point types. *enumlist* is defined as (definition from *include/src/dictionary.h*):
	
	`typedef struct`
	`{`
	`    int    value;`
	`    char*  text;`
	`} enumlist;`

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 120 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 140 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 205

“hsc_param_type()” on page 174

“hsc_enumlist_destroy()” on page 140

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
number	(in) point type number
name	(out) point type name
namelen	(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 120 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);
    pnttypp++;
}
```

See also

“hsc_pnttyp_number()” on page 191

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 120 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 203

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
point	(in) point number
name	(out) entity name
namelen	(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 120 will retrieve the error code.

hsc_point_fullname()

Returns the full item 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
point	(in) point number
name	(out) full item name
namelen	(in) length of name string

Description

This function takes a point number and return the point's full item 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 120 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
point	(in) point number
count	(out) number of children
children	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned and calling “c_geterrno()” on page 120 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
point	(in) point number
piNumAncestors	(out) number of ancestors
ppAncestors	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned and calling “c_geterrno()” on page 120 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
parent	(in) point number of the parent
piNumChildren	(out) number of children
ppChildren	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned and calling “c_geterrno()” on page 120 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_descendents()

Returns all containment descendents below a specified point.

C/C++ synopsis

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

int hsc_point_get_containment_descendents(
    PNTNUM   point,
    int*     piNumDescendents,
    PNTNUM** ppDescendents
);
```

Arguments

Argument	Description
point	(in) point number
piNumDescendents	(out) number of descendents
ppDescendents	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned and calling “c_geterrno()” on page 120 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
child	(in) point number of the child point
piNumParents	(out) number of parents
ppParents	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned and calling “c_geterrno()” on page 120 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

Argument	Description
point	(in) point number
count	(out) number of parents
parents	(out) array of parents

Description

Returns all parents, both containment and reference.

The array must be cleared by calling “hsc_em_FreePointList()” on page 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned and calling “c_geterrno()” on page 120 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
point	(in) point number
piNumRefItems	(out) number of references
ppRefItems	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned and calling “c_geterrno()” on page 120 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
point	(in) point number
piNumRefItems	(out) number of referers
ppRefItems	(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 135.

Diagnostics

If successful, *0* is returned, otherwise *-1* is returned and calling “c_geterrno()” on page 120 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
point	(in) point number
gGUID	(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 120 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
point	(in) point number
name	(out) parameter name
namelen	(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 120.

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()” on page 204

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 203

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
point	(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 120 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
file	(in) server file number.
delay	(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 120 will retrieve one of the following error codes:

[FILLCK]	File locked to another task
[RECLCK]	Record locked to another task
[DIRLCK]	File's directory locked to another task
[BADFIL]	Illegal file number specified

See also

"hsc_lock_record()" on page 155

"hsc_unlock_file()" on page 207

"hsc_unlock_record()" on page 208

Related topics

"Ensuring database consistency" on page 53

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
file	(in) server file number
record	(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 120 will retrieve one of the following error codes:

[FILLCK]	File locked to another task
[RECLCK]	Record locked to another task
[DIRLCK]	File's directory locked to another task
[BADFIL]	Illegal file number specified
[BADRECD]	Illegal record number specified

See also

“hsc_lock_file()” on page 154

“hsc_unlock_file()” on page 207

“hsc_unlock_record()” on page 208

Related topics

“Ensuring database consistency” on page 53

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 23

"isinfdouble()" on page 219

"inffloat()" on page 212

"isinffloat()" on page 220

"Data types" on page 29

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 23

"isinffloat()" on page 220

"infdouble()" on page 211

"isinfdouble()" on page 219

"Data types" on page 29

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
pvvalue	(in) A pointer to a *PARvalue* structure.
int2_val	(in) The *int2* value to insert into the *PARvalue* 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 142 and "hsc_insert_attrib_byindex()" on page 149 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 120 will retrieve the following error code:

BUFFER_TOO_SMALL	The pointer to the *PARvalue* is invalid, that is, null.

Example

See the examples in "hsc_insert_attrib()" on page 142 and "hsc_insert_attrib_byindex()" on page 149.

See also

"DbletoPV()" on page 112

"hsc_insert_attrib()" on page 142

"hsc_insert_attrib_byindex()" on page 149

"Int4toPV()" on page 214

"PritoPV()" on page 234

"RealtoPV()" on page 236

"StrtoPV()" on page 245

"TimetoPV()" on page 246

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
pvvalue	(in) A pointer to a *PARvalue* structure.
int4_val	(in) The *int4* value to insert into the *PARvalue* 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 142 and “hsc_insert_attrib_byindex()” on page 149 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 120 will retrieve the following error code:

BUFFER_TOO_SMALL	The pointer to the *PARvalue* is invalid, that is, null.

Example

See the examples in “hsc_insert_attrib()” on page 142 and “hsc_insert_attrib_byindex()” on page 149.

See also

“DbletoPV()” on page 112

“hsc_insert_attrib()” on page 142

“hsc_insert_attrib_byindex()” on page 149

“Int2toPV()” on page 213

“PritoPV()” on page 234

“RealtoPV()” on page 236

“StrtoPV()” on page 245

“TimetoPV()” on page 246

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
intbuf	(in) source integer buffer containing ASCII.
intbuflen	(in) size of source integer buffer in bytes.
chrbuf	(out) destination character buffer.
chrbuflen	(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 102

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
pGdaError	(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 120 will retrieve the error code.

Diagnostics

Returns *TRUE* if *pGdaError* indicates an error condition, otherwise it returns *FALSE*.

See also

"IsGDAwarning()" on page 218

"IsGDAnoerror()" on page 217

"hsc_IsError()" on page 152

"c_geterrno()" on page 120

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
pGdaError	(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 120 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 page 218

“IsGDAerror()” on page 216

“c_geterrno()” on page 120

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
pGdaError	(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 120 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 216

“IsGDAnoerror()” on page 217

“hsc_IsWarning()” on page 153

“c_geterrno()” on page 120

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 23
"infdouble()" on page 211
"inffloat()" on page 212
"isinffloat()" on page 220
"Data types" on page 29

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 23

"inffloat()" on page 212

"infdouble()" on page 211

"isinfdouble()" on page 219

"Data types" on page 29

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
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 23

"nandouble()" on page 226

"nanfloat()" on page 227

"isnanfloat()" on page 222

"Data types" on page 29

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 23

"nanfloat()" on page 227

"nandouble()" on page 226

"isnandouble()" on page 221

"Data types" on page 29

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
year	(in/out) number of years since 0 AD (for example, 2012).
month	(in/out) month (1–12).
day	(in/out) day (1–31).
julian	(in) number of Julian days.

Description

Converts between Julian days (used by the History subsystem) and a Gregorian date (day, month, year format).

c_gtoj	converts from a Gregorian date to Julian days.
c_jtog	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 121

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
progname	(in) name of program module
lineno	(in) line number in program module
format	(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
value	(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 23

"isnandouble()" on page 221

"nanfloat()" on page 227

"isnanfloat()" on page 222

"Data types" on page 29

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 23
"isnanfloat()" on page 222
"nandouble()" on page 226
"isnandouble()" on page 221
"Data types" on page 29

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*  prmblk
);
```

Arguments

Argument	Description
crt	(in) Station number.
message	(in) pointer to null-terminated string to be sent to the Station.
param1	(in) value of parameter 1 required to identify response task request.
prmblk	(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.

c_oprstr_info	outputs information only messages.
c_oprstr_message	outputs an invalid request message that is cleared after a TIME_RQUEST seconds. This constant is defined in 'server/src/system'.
c_oprstr_prompt	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, the 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 c_oprstr_response to fetch the response.
c_oprstr_response	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 120 will retrieve the following error code:

[M4_INVALID_CRT]	An invalid CRT was specified.

Warnings

c_oprstr_prompt requires that the calling task “c_trm04()” on page 249 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 248 function. If a response has not been received from the operator after an interval (specified by you in the “c_tmstrt_...()” on page 248 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 247.

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
point	(in) Point type/number to be processed.
param	(in) Point parameter to be processed. -1 for all parameters.
status	(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 page 231

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
point	(in) point type/number to be processed.
param	(in) point parameter to be processed. -1 for all parameters.
status	(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 230

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
point	(in) Point type/number to be processed.
param	(in) Point parameter to be processed. -1 for all parameters.
value	(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 120 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_2()" on page 233

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
point	(in) Point type/number to be processed.
param	(in) Point parameter to be processed. -1 for all parameters.
value	(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 120 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 232

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
pvvalue	(in) A pointer to a *PARvalue* structure.
priority	(in) The priority value to insert into the *PARvalue* structure.
sub-priority	(in) The sub-priority value to insert into the *PARvalue* 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 142 and "hsc_insert_attrib_byindex()" on page 149 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 120 will retrieve the following error code:

BUFFER_TOO_SMALL	The pointer to the *PARvalue* is invalid, that is, null.

Example

See the examples in "hsc_insert_attrib()" on page 142 and "hsc_insert_attrib_byindex()" on page 149.

See also

"DbletoPV()" on page 112

"hsc_insert_attrib()" on page 142

"hsc_insert_attrib_byindex()" on page 149

"Int2toPV()" on page 213

"Int4toPV()" on page 214

"RealtoPV()" on page 236

"StrtoPV()" on page 245

"TimetoPV()" on page 246

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
crt	(in) CRT number.
printer	(in) printer number.
filename	(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 120 will retrieve the following error code:

[M4_QEMPTY]	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 *progname='sample.c';

/* send the sample file to the printer */
if (c_prsend_printer(PRINTER_NO, SAMPLE_FILENAME) == -1)
{
    c_logmsg(progname, '123', 'c_prsend_printer
    error 0x%x', c_geterrno());
    exit(ierr);
}
```

RealtoPV()

Inserts a real value into a *PARvalue* union.

C/C++ synopsis

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

PARvalue* RealtoPV(
    float       real_val,
    PARvalue*   pvvalue
);
```

Arguments

Argument	Description
pvvalue	(in) A pointer to a *PARvalue* structure.
real_val	(in) The real value to insert into the *PARvalue* 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 142 and "hsc_insert_attrib_byindex()" on page 149 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 120 will retrieve the following error code:

BUFFER_TOO_SMALL	The pointer to the *PARvalue* is invalid, that is, null.

Example

See the examples in "hsc_insert_attrib()" on page 142 and "hsc_insert_attrib_byindex()" on page 149.

See also

"DbletoPV()" on page 112

"hsc_insert_attrib()" on page 142

"hsc_insert_attrib_byindex()" on page 149

"Int2toPV()" on page 213

"Int4toPV()" on page 214

"PritoPV()" on page 234

"StrtoPV()" on page 245

"TimetoPV()" on page 246

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
lrn	(in) Logical resource number of the task to be requested.
prmblk	(in) pointer to parameter block to be passed to the task.

Description

c_rqtskb	requests the specified queued task without any parameters. If the task is already running, no action is taken.

c_rqtskb_prm	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.
	
	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 trbtbl_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:
	
	`prm my_pblk;`
	`int myLrn;`
	`myLRN = 111; //user application LRN`
	`...`
	`return_val = c_rqtskb_prm(myLrn, (int2*) &my_pblk);`
	
	Note that the some LRNs listed in the *def/src/lrns* 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.
	
	Stations 21-40, if assigned, use other LRNs. These can be displayed using the utility *usrlrn* with the options *-p -a*.

Notes

To call up a named display in Station you need to:

1. Memset to 0 the Parameter Request Block structure.
2. Ensure the prmblk.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(&prmblk, 0, sizeof(prmblk));
prmblk.crt = Station Number (in this test case it is 1)
prmblk.param1 = 1;
prmblk.param2 = 0;
prmblk.path is set via c_chrint("testdisplay",11,prmblk.path,sizeof(prmblk.path));

iDisplayLRN = dsply_lrn(station number);
c_rqtskb_prm(iDisplayLRN,(int2 *)&prmblk);
```

To call up a numbered display:

```
prmblk.crt = Station Number (in this test case it is 1)
prmblk.param1 = 1;
prmblk.param2 = 14;

iDisplayLRN = dsply_lrn(station number);
c_rqtskb_prm(iDisplayLRN,(int2 *)&prmblk);
```

Diagnostics

Upon successful completion a value of *0* is returned. Otherwise, *-1* is returned and calling "c_geterrno()" on page 120 will retrieve one of the following error codes:

[M4_ILLEGAL_LRN]	An illegal LRN has been specified or the task does not exist.
[M4_BUSY_TRB]	The requested tasks request block is busy, could not pass parameters.
[INVALID_SEMVAL]	The requested task has too many outstanding requests.

See also

"c_getreq()" on page 128

"c_tstskb()" on page 251

"c_wttskb()" on page 255

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

Argument	Description
point	(in) Point type/number to be processed.
param	(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 120 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_2()" on page 240

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
point	(in) Point type/number to be processed.
param	(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 120 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 239

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
point	(in) point type and/or number to be processed
param	(in) point parameter to be processed. PV=0 through A4=7
value	(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_2()" on page 242

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
point	(in) point type and/or number to be processed.
param	(in) point parameter to be processed. PV=0 through A4=7.
value	(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 241

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
rtu	(in) the controller number to be updated
seconds	(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 113

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
pvvalue	(in) A pointer to a *PARvalue* structure
string_val	(in) The character string to insert into the *PARvalue* 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 142 and "hsc_insert_attrib_byindex()" on page 149 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 120 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 142 and "hsc_insert_attrib_byindex()" on page 149.

See also

"DbletoPV()" on page 112

"hsc_insert_attrib()" on page 142

"hsc_insert_attrib_byindex()" on page 149

"Int2toPV()" on page 213

"Int4toPV()" on page 214

"PritoPV()" on page 234

"RealtoPV()" on page 236

"TimetoPV()" on page 246

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
pvvalue	(in) A pointer to a *PARvalue* structure
time_val	(in) The time value to insert into the *PARvalue* 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 142 and “hsc_insert_attrib_byindex()” on page 149 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 120 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 142 and “hsc_insert_attrib_byindex()” on page 149.

See also

“DbletoPV()” on page 112

“hsc_insert_attrib()” on page 142

“hsc_insert_attrib_byindex()” on page 149

“Int2toPV()” on page 213

“Int4toPV()” on page 214

“PritoPV()” on page 234

“RealtoPV()” on page 236

“StrtoPV()” on page 245

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
tmridx	(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_...()” on page 248.

See also

“c_tmstrt_...()” on page 248

“c_trmtsk()” on page 250

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
cycle	(in) time interval between executions in seconds.
param1	(in) parameter passed to task as parameter 1.
param2	(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 237 every interval. A timer is stopped by calling “c_tmstop()” on page 247.

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 128.

c_tmstrt_single	requests the specified task only once.
c_tmstrt_cycle	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 247

“c_getreq()” on page 128

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
status	(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 page 249. 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 255 calls.

If the task has been marked for deletion via a "c_deltsk()" on page 111 call then the task will be removed from the system.

See also

"c_rqtskb...()" on page 237

"c_tstskb()" on page 251

"c_wttskb()" on page 255

"c_deltsk()" on page 111

c_trmtsk()

Terminates a task with error status.

C/C++ synopsis

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

void __stdcall c_trmtsk(
    int2    status
);
```

Arguments

Argument	Description
status	(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 255 calls.

If the task has been marked for deletion via a “c_deltsk()” on page 111 call then the task will be removed from the system.

See also

“c_trm04()” on page 249

“c_deltsk()” on page 111

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
lrn	(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 120 will retrieve one of the following error codes:

[M4_ILLEGAL_LRN]	An illegal LRN has been specified.
[M4_BUSY_TRB]	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 255

“c_rqtskb...()” on page 237

“c_trm04()” on page 249

“c_trmtsk()” on page 250

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
string	(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 102

“c_intchr()” on page 215

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
wdtidx	(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()” on page 254.

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()” on page 254

Related topics

“Monitoring the activity of a task” on page 45

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
timeout	(in) time interval before watchdog timer takes action indicated by mode.
mode	(in) mode of action to be taken by the watchdog timer: • *WDT_MONITOR* monitor timer entry only. • *WDT_ALARM_ONCE* generate an alarm on first failure only. • *WDT_ALARM* generate an alarm on each failure. • *WDT_RESTART_TASK* restart task on first failure, reboot the server system on second failure. • *WDT_RESTART_SYS* restart the server system on failure.

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 page 253 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 page 254 is unable to create a timer, it returns a *0*.

See also

“c_wdon()” on page 253

“c_wdstrt()” on page 254

Related topics

“Monitoring the activity of a task” on page 45

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
lrn	(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:

[M4_ILLEGAL_LRN]	An illegal LRN has been specified.

See also

"c_tstskb()" on page 251

"c_rqtskb...()" on page 237

"c_trm04()" on page 249

"c_trmtsk()" on page 250

Backward-compatible functions

The following functions are available for backwards compatibility.

"c_badpar()" on page 257
"c_gethstpar_...()" on page 258
"c_pps()" on page 261
"c_ppsw()" on page 262
"c_ppv()" on page 263
"c_ppvw()" on page 264
"c_sps()" on page 265
"c_spsw()" on page 266
"c_spv()" on page 267
"c_spvw()" on page 268

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 PKS server release R400 or later *c_badpar()* will only be able to access points in the range: 1<= point number <= 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

Argument	Description
point	(in) point type/number to be tested.
param	(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 225

“hsc_param_value()” on page 176

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 PKS server release R400 or later *c_gethstpar()* will only be able to access points in the range: 1<= point number <= 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
type	(in) history type (see Description).
date	(in) start date of history to retrieve in Julian days (number of days since 1 Jan 1981).
time	(in) start time of history to retrieve in seconds since midnight.
offset	(in) offset from latest history value in history intervals (where offset=1 is the most recent history value).
numhst	(in) number of history values to be returned per Point.
points	(in) array of Point type/numbers to process (maximum of 100 elements).
params	(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 *def* folder (maximum 100 elements).
numpnt	(in) number of Points to be processed.

Argument	Description
archive	(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 *<server folder>\archive*. For example, to access the files in *<server folder>\archive\ay2012m09d26h11r008*, the archive argument is *ay2012m09d26h11r008*.
values	(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 *numpnt* * *numhst*.

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.

c_gethstpar_date	retrieves history values from a specified date and time.
c_gethstpar_ofst	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 y^{th} history value for the x^{th} point.

The history type is specified by 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

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_ILLEGAL_VAL]	Illegal number of Points or history values specified.
[M4_ILLEGAL_HST]	Illegal history type or interval specified.
[M4_VAL_NOT_FND]	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 179

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 PKS 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
point	(in) Point type/number to be processed.
param	(in) Point parameter to be processed. -1 for all parameters.
status	(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()" on page 262

"c_pps_2()" on page 230

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 PKS 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
point	(in) point type/number to be processed.
param	(in) point parameter to be processed. -1 for all parameters.
status	(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 261

"c_ppsw_2()" on page 231

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 PKS 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
point	(in) Point type/number to be processed.
param	(in) Point parameter to be processed. -1 for all parameters.
value	(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 120 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()" on page 264

"c_ppv_2()" on page 232

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 PKS server release R400 or later *c_ppvw()* will only be able to access points in the range: 1<= point number <= 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
point	(in) Point type/number to be processed.
param	(in) Point parameter to be processed. -1 for all parameters.
value	(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 120 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()” on page 263

“c_ppvw_2()” on page 233

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 PKS 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
point	(in) Point type/number to be processed.
param	(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 120 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()” on page 266

“c_sps_2()” on page 239

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 PKS 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
point	(in) Point type/number to be processed.
param	(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 120 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 265

“c_spsw_2()” on page 240

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 PKS 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
point	(in) point type and/or number to be processed
param	(in) point parameter to be processed. PV=0 through A4=7
value	(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()” on page 268

“c_spv_2()” on page 241

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 PKS 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
point	(in) point type and/or number to be processed.
param	(in) point parameter to be processed. PV=0 through A4=7.
value	(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 267

"c_spvw_2()" on page 242

Examples

There are several examples located under the server install folder in *user/examples/src*. These can be used as a basis for your own programs.

The examples are:

Example	Description
test1.c	a very simple task which prints 'Hello World' every time it is requested.
test2.c	a simple task that generates 3 alarms when requested.
test3.c	a task that demonstrates dealing with points.
test4.c	a simple task that demonstrates the use of user tables.
test5.c	a more completed task that demonstrates the use of user tables.
test6.c	a utility that demonstrates dealing with points.
test7.c	a complicated task demonstrating the use of watchdog timers, scan point special, *hsc_param_values* and *hsc_param_value_put*.
test8.c	a simple task that shows the information passed on the prmblk when the task is requested.

Network API reference

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

Related topics

"Prerequisites" on page 272

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 *Server and Client Configuration Guide*.
- Be familiar with user access and file management as described in the *System Administration Guide*.

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

Related topics

"About the Network API" on page 274
"Summary of Network API functions" on page 276
"Using the Network API" on page 277
"Using Microsoft Visual Studio or Visual Basic to develop Network API applications" on page 285
"Folder structures for C/C++ applications" on page 288
"Network API applications fail to run" on page 289

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.
- **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.

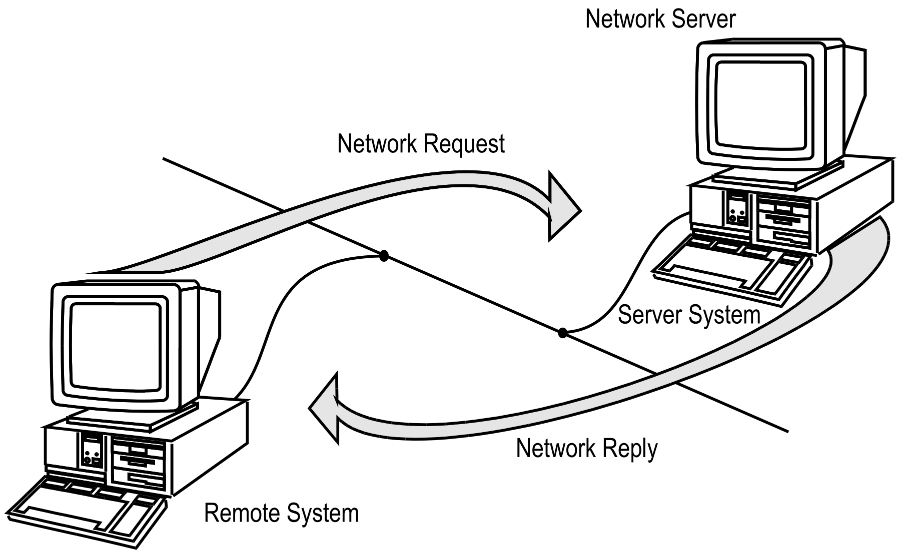


Figure 9: Network server

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 (*hsserva*, *hsservb*)
Single server, dual network	Basename appended with 0 or 1 (*hsserv0*, *hsserv1*)
Redundant server, dual network	Basename appended with a or b and 0 or 1 (*servera0*, *servera1*, *serverb0*, *serverb1*)

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
Generate Alarms and Events	
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.
Point Parameter Access	
rhsc_point_numbers_2	Convert a point name into an internal point number that is used in other calls such as rhsc_param_numbers_2 and rhsc_param_values_2.
rhsc_param_numbers_2	Convert a parameter name into an internal parameter number that is used in other calls such as rhsc_param_values_2 or rhsc_param_value_puts_2.
rhsc_param_values_2	Read a list of point parameter values from the database.
rhsc_param_value_bynames	Similar to rhsc_param_values_2 but provides a simpler calling interface but is less efficient.
rhsc_param_value_puts_2	Write a list of point parameter values to the database.
rhsc_param_value_put_bynames	Similar to rhsc_param_value_puts_2 but provides a simpler calling interface but is less efficient. **rgetpnt**, **rgetval** and **rputval** have been included for backwards compatibility.
Historical Information Access	
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.
User File Information Access	
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.
Error String Lookup	
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_napierrstr	Look up the error string for an error number.

Using the Network API

To:	Go to:
Determine the point numbers	"Determining point numbers" on page 277
Determine the parameter numbers	"Determining parameter numbers" on page 278
Access point parameters	"Accessing point parameters" on page 278
Access historical information	"Accessing historical information" on page 279
Access user tables	"Accessing user table data" on page 280
Look up error strings	"Looking up error strings" on page 283
Access parameter values by name	"Functions for accessing parameter values by name" on page 283
Learn about Visual Basic's migration requirements	"Migration requirements for PlantScape pre-R500 Visual Basic applications"

Related topics

"Determining point numbers" on page 277
"Determining parameter numbers" on page 278
"Accessing point parameters" on page 278
"Accessing historical information" on page 279
"Accessing user table data" on page 280
"Looking up error strings" on page 283
"Functions for accessing parameter values by name" on page 283

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\Napitst* 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 306
- “rhsc_param_value_put_bynames” on page 309
- “rhsc_param_hist_date_bynames” on page 297
- “rhsc_param_hist_offset_bynames” on page 297

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\Napitst* 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\Mfcnetapitest* 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 306
- “rhsc_param_value_put_bynames” on page 309
- “rhsc_param_hist_date_bynames” on page 297
- “rhsc_param_hist_offset_bynames” on page 297

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) 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 306 and “rhsc_param_value_put_bynames” on page 309 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\Napitst* 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.

User table point number storage

From Experion PKS R400, point numbers require 32 bits of data for storage; previously they required 16 bits. Therefore, when configuring new User Tables that contain point numbers, 32 bits of storage should be allocated. When upgrading from a previous version of Experion PKS, point numbers will have most likely been stored using 16 bits of storage. The User Table will need to be reconfigured to use the required 32 bits of storage. Client applications will also have to be updated to access the user table appropriately.

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\Napitst* 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\Vbnetapitst* 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.tmout = 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 hexidecimal 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 .

If you are developing Network API applications in Visual Basic, you need Microsoft Visual Basic Version 6 SP5.

Setting up Microsoft Visual Studio for Network API applications

Setting up Microsoft Visual Studio involves:

- Creating a project workspace.
- Modifying the `Include Directories` and `Library Directories` for the project to include the Experion folders.
- Modifying the project settings for Experion application development.

To create a project workspace in Visual Studio

1. In the Microsoft Visual Studio application window, choose **File** > **New** > **Project**.
   The **New Project** window appears.
2. In the hierarchical list, expand **Installed**, expand **Templates**, and then click **Visual C++** .
3. At the top of the window, select the supported version of the .NET Framework.
4. Select the project type you want to develop (**Win 32 Console Application**, **MFC Application**, and so on).
5. Complete the **Name**, **Location**, and **Solution name**, and other details for the project.
6. Click **OK** to create the project.
   The **Application Wizard** appears.
7. If required, review and modify the **Application Settings**.
8. Click **Finish**.

To modify the Include Directories and Library Directories for the project

1. In the Microsoft Visual Studio application window, choose **Project** > *name* **Properties**, where *name* is the project name. If you have the *name* (the project name) item selected in the Solution Explorer, choose **Project** > **Properties**.
   The **Property Pages** window appears.
2. In the hierarchical list, expand **Configuration Properties**, and then click **VC++ Directories**.
3. Click **Include Directories**.
4. On the right-side of the **Include Directories** row, click the drop-down arrow and then click **<Edit...>**.
   The **Include Directories** dialog box appears.
5. Click the **New Line** icon, then click the browse icon and select the following folder:
   `<install folder>\Honeywell\Experion PKS\Client\NetworkAPI\include`
   Where `<install folder>` is the location where Experion is installed.
6. Click **OK**.
7. Click **Library Directories**.
8. On the right-side of the **Library Directories** row, click the drop-down arrow and then click **<Edit...>**.
   The **Library Directories** dialog box appears.
9. Click the **New Line** icon, then click the browse icon and select the following folder:
   `<install folder>\Honeywell\Experion PKS\Client\NetworkAPI\lib`

Where *<install folder>* is the location where Experion is installed.

10 Click **OK**.

11 In the **Property Pages** window, click **OK**.

To modify the project settings

1 In the Microsoft Visual Studio application window, choose **Project** > *name* **Properties**, where *name* is the project name. If you have the *name* (the project name) item selected in the Solution Explorer, choose **Project** > **Properties**.

The **Property Pages** window appears.

2 In the hierarchical list, expand **Configuration Properties**, expand **C/C++**, and then click **Preprocessor**.

3 Add **NT** to the **Preprocessor Definitions**.

4 In the hierarchical list, click **Code Generation**.

If this is not visible, expand **Configuration Properties** and then expand **C/C++**.

5 Click **Runtime Library**.

6 On the right-side of the **Runtime Library** row, click the drop-down arrow and then select **Multi-threaded DLL (/MD)**.

For debugging, use **Multi-threaded Debug DLL (/MDd)**. Use **Multi-threaded DLL (/MD)** only for release compiles. If you use the incorrect library for a debug compile, the error code does not propagate correctly (it will be always zero).

7 In the hierarchical list, click **General**.

If this is not visible, expand **Configuration Properties** and then expand **C/C++**.

8 Click **Additional Include Directories**.

9 On the right-side of the **Additional Include Directories** row, click the drop-down arrow and then click **<Edit...>**.

The **Additional Include Directories** dialog box appears.

10 Click the **New Line** icon, then click the browse icon and select the following folder:

<install folder>\Honeywell\Experion PKS\Client\NetworkAPI\include

Where *<install folder>* is the location where Experion is installed.

11 Click **OK**.

12 In the hierarchical list, expand **Configuration Properties**, expand **Linker**, and then click **Input**.

13 Click **Additional Dependencies**.

14 On the right-side of the **Additional Dependencies** row, click the drop-down arrow and then click **<Edit...>**.

The **Additional Dependencies** dialog box appears.

15 Type `hscnetapi.lib`, and then click **OK**.

16 Click **Ignore All Default Libraries**.

17 On the right-side of the **Ignore All Default Libraries** row, click the drop-down arrow and then select **No**.

18 If you are developing an MFC application and want to dynamically link to MFC, complete the following steps:

a In the hierarchical list, expand **Configuration Properties**, and then click **General**.

b Click **Use of MFC**.

c On the right-side of the **Use of MFC** row, click the drop-down arrow and then select **Use MFC in a Shared DLL**.

d In the hierarchical list, expand **Configuration Properties**, expand **Linker**, and then click **Input**.

e In the **Ignore Specific Default Libraries** row, .ensure that *msvcrt* is *not* listed.

19 If you are developing an MFC application and want to statically link to MFC, complete the following steps:

a In the hierarchical list, expand **Configuration Properties**, and then click **General**.

b Click **Use of MFC**.

c On the right-side of the **Use of MFC** row, click the drop-down arrow and then select **Use MFC in a Static Library**.

d In the hierarchical list, expand **Configuration Properties**, expand **Linker**, and then click **Input**.

e Click **Ignore Specific Default Libraries**.

f On the right-side of the **Ignore Specific Default Libraries** row, click the drop-down arrow and then click **<Edit...>**.

The **Ignore Specific Default Libraries** dialog box appears.

g Type `msvcrt`, and then click **OK**.

20 Click **OK** to save your project settings.

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 **Project** > **References**.

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 2008 SP1, 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

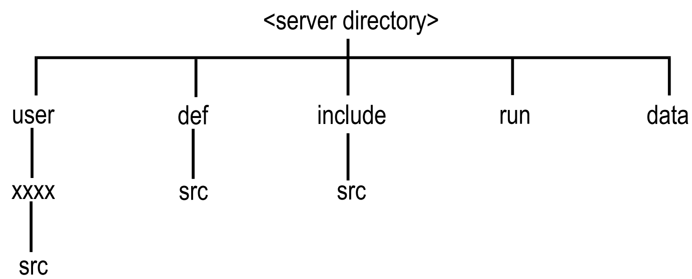
The structure shown below should be used for development of C/C++ Server API applications which will run on the server.

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 *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 *data* folder contains all server database files.



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 PKS NetAPI installed, the application will fail to run. The **hscnetapi.dll** that is shipped with Experion PKS NetAPI is required for all NetAPI applications to run on an Experion PKS server.

To resolve this problem, copy the file `%windir%\System32\hscnetapi.dll` from a node that has the Experion PKS NetAPI installed and copy it into the same directory on the computer that doesn't have Experion PKS NetAPI installed and needs to run the application.

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" on page 292
- "Backward-compatibility Functions" on page 323
- "Diagnostics for Network API functions" on page 344

Functions

This section contains Network API functions.

See also

"Backward-compatibility Functions" on page 323

"Diagnostics for Network API functions" on page 344

Related topics

"hsc_bad_value" on page 292
"hsc_napierrstr" on page 293
"rgetdat" on page 293
"rhsc_notifications" on page 295
"rhsc_param_hist_date_bynames" on page 297
"rhsc_param_hist_offset_bynames" on page 297
"rhsc_param_hist_dates_2" on page 301
"rhsc_param_hist_offsets_2" on page 302
"rhsc_param_numbers_2" on page 304
"rhsc_param_value_bynames" on page 306
"rhsc_param_value_put_bynames" on page 309
"rhsc_param_value_put_sec_bynames" on page 311
"rhsc_param_value_puts_2" on page 313
"rhsc_param_values_2" on page 316
"rhsc_point_numbers_2" on page 319
"rputdat" on page 321

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
server	(in) Name of server that the database resides on.

Argument	Description
num_points	(in) The number of points passed to *rgetdat_xxxx* in the *getdat_xxxx_data*argument.
getdat_xxxx_data	(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 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:

int2 type	(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: RGETDAT_TYPE_INT2, RGETDAT_TYPE_INT4, RGETDAT_TYPE_REAL4, RGETDAT_TYPE_REAL8, RGETDAT_TYPE_STR, RGETDAT_TYPE_BITS
int2 file	(in) Absolute database file number to retrieve/store field.
int2 rec	(in) Record number in above file to retrieve/store field.
int2 word	(in) Word offset in above record to retrieve/store field.
int2 start_bit	(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.
int2 length	(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.
int2 flags	(in) Specifies the direction to read/write for circular files. (0: newest record, 1: oldest record)
rgetdat_value value	(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.
int2 status	(out) return value of actual remote getdat/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, are defined as follows:
short int2	Two byte signed integer.
long int4	Four byte signed integer.
float real4	Four byte IEEE floating point number.
double real	Eight byte (double precision) IEEE floating point number.
char str*	Pointer to string. (Note allocation of space for retrieving a string is the responsibility of the program calling *rgetdat*, see *rgetdat_data* structure description above).

unsigned short bits	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 344.

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
szHostname	(in) Name of server that the data resides on (that is, server hostname)
cprmbd	(in) Number of notifications requested
notd	(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:

struct timeb timebuf	(in) reserved for future use
n_long nPriority	(in) priority
n_long nSubPriority	(in) sub priority
n_char szName*	(in) name (usually pnt name)
n_char szEvent*	(in) event (eg. RCHANGE)
n_char szAction*	(in) action (eg. OK, ACK, CNF)
n_char szLevel*	(in) level (eg. CB, MsEDE, NAPI)
n_char szDesc*	(in) description (usually param name)
n_char szValue*	(in) alarm value
n_char szUnits*	(in) alarm units
n_long fStatus	(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[] = {{'pntana1','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 pntana1.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 = ActionStrings0];          /* 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 344.

See also

“hsc_notif_send()” on page 156

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
szHostName	(in) Name of server on which the data resides
cHstRequests	(in) Number of rghstbd elements
rghstbd	(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:

n_long dtStartDate	(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.
n_float tmStartTime,	(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.
n_long nHstOffset,	(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.
n_long fGetHstParStatus,	(out) The status returned by the gethstpar function.
n_short nHstType,	(in) The type of history to retrieve (See Description).
n_ushort cPntPrmNames,	(in) The number of point / parameter pairs requested.
n_ushort cHstValues	(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.
n_char szArchivePath,*	This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string.
*n_char** rgszPointNames,*	(in) An array of point names to process.
*n_char** rgszParamNames*	(in) An array of parameter names to process. Each parameter is associated with the corresponding entry in the rgszPointNames array.
n_long rgfPntPrmStatus*	(out) The status returned by the Server when calling *hsc_point_number* and *hsc_param_number* for each point parameter pair.
n_float rgnHstValues*	(out) A pointer to a 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

Value	Description
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*cHstValues*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 * number of history requests) + (2 * 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 * number of history requests) + (2 * number of point parameter pairs) + sum of string lengths
of point names, parameter names and archive paths in bytes < 4000
```

- For response packets:

```
(4 * number of history requests)+ (4 * (For each history request the sum of (cPntPrmNames +
cPntPrmNames * cHstValues))) < 4000
```

Diagnostics

See “Diagnostics for Network API functions” on page 344.

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].cHstValues * 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
server	(in) Name of server that the database resides on
num_requests	(in) The number of history requests
gethstpar_x_data	(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:

n_ushort hist_type	(in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following: HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE
n_ulong hist_start_date	(in) Start date of history to receive in Julian days (number of days since 1st January 1981).
n_ufloat hist_start_time	(in) Start time of history to retrieve in seconds since midnight.
n_ushort num_hist	(in) Number of history values per point to be retrieved.
n_ushort num_points	(in) Number of points to be processed. MAXIMUM value allowed is 20.
n_ulong point_type_nums*	(in) Array (of size *num_points*) containing the point type/numbers of the point history values to retrieve.
n_ushort point_params*	(in) Array of (of size *num_points*) containing the parameter numbers of the history values to retrive.
n_char archive path*	(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.
n_ufloat hist_values*	(out) Array (of size *num_points* * *num_hist*) to provide storage for the returned history values.
n_ushort gethst_status	(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:

n_ushort hist_type	(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
n_ushort hist_offset	(in) Offset from latest history value in history intervals where offset=1 is the most recent history value).
n_ushort num_hist	(in) Number of history values per point to be retrieved.
n_ushort num_points	(in) Number of points to be processed. MAXIMUM value allowed is 20.
n_ulong point_type_nums*	(in) Array (of size *num_points*) containing the point type/numbers of the point history values to retrieve.
n_ushort point_params*	(in) Array (of size *num_points*) containing the parameter numbers of the history values to retrieve.
n_char archive path*	(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.
n_float hist_values*	(out) Array (of size *num_points* * *num_hist*) to provide storage for the returned history values.
n_ushort gethst_status	(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 * number of rgethstpar_date_data_2 structs) + (6 * combined point / param pair count across all
    rgethstpar_date_data_2 structs) +  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)) + (strings ?) < 4000

- For request packets for rhsc_param_hist_offsets_2:

```
(12 * number of rgethstpar_ofst_data_2 structs) + (6 * combined point / param pair count across all
    rgethstpar_ofst_data_2 structs) + 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)) + (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 344.

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
szHostname	(in) Name of server on which the database resides
cprmnd	(in) The number of parameter name resolutions requested
rgprmnd2	(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:

n_ulong nPnt	(in) point number
n_char szPrmName*	(in) parameter name to resolve
n_ushort nPrm	(out) parameter number returned
n_long fStatus	(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 344.

See also

“rhsc_point_numbers_2” on page 319

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
szHostname	(in) Name of server that the data resides on.
nPeriod	(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.
cprmbd	(in) Number of parameter values requested.
rgprmbd	(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:

n_char szPntName*	in) point name
n_char szPrmName*	(in) parameter name
n_long nPrmOffset	(in) parameter offset
PARvalue pupvValue*	(out) parameter value union
n_ushort nType	(out) value type

n_long fStatus	(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 * number of point parameters) + sum of string lengths of point names + sum of string lengths
  of parameter names < 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:

  ```
  (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_DBLE:
                    printf('%s.%s has the value %f\n',
                        rgprmbd[].szPntName,
                        rgprmbd[i].szPrmName,
                        rgprmbd[i].pupvValue->dble);
                    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 344.

See also

“rhsc_param_value_puts_2” on page 313

“rhsc_param_value_put_bynames” on page 309

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
szHostname	(in) Name of server that the data resides on
cprmbd	(in) Number of controls to parameters values requested
rgprmbd	(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:

n_char szPntName*	(in) point name
n_char szPrmName*	(in) parameter name
n_long nPrmOffset	(in) parameter offset
PARvalue pupvValue*	(in) parameter value union
n_ushort nType	(in) value type
n_long fStatus	(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 344) 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 enumeration 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[] = {{'pntana1','SP', 1},{'pntana1', 'DESC', 1}};
#define cprmbd (sizeof(rgprmbd)/sizeof(PARAM_BYNAME_DATA))

/* Allocate space and set the value and type to control pntana1.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 pntana1.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 344.

See also

“rhsc_param_values_2” on page 316

“rhsc_param_value_put_bynames” on page 309

rhsc_param_value_put_sec_bynames

This function acts similarly to “rhsc_param_value_put_bynames” on page 309, 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
szHostname	(in) Name of server that the data resides on
cprmbd	(in) Number of controls to parameters values requested
rgprmbd	(in/out) Pointer to an array of PARAM_BYNAME_DATA structures (one array element for each request)
nStn	(in) Station number, which the Network Server uses in the associated CHANGE event for this call. If operator-based security is used, the operator's name/ID will also be captured. 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 344.

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
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
szHostname	(in) Name of server that the database resides on
cprmvd	(in) The number of controls to parameters requested
rgprmvd2	(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:

n_ulong nPnt	(in) point number
n_ushort nPrm	(in) parameter number
n_long nPrmOffset	(in) point parameter offset
PARvalue pupvValue*	(in) parameter value union
n_ushort nType	(in) value type
n_long fStatus	(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 344.

See also

“rhsc_param_values_2” on page 316

“rhsc_param_value_put_bynames” on page 309

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
szHostname	(in) Name of server that the database resides on.

Argument	Description
nPeriod	(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.
cprmvd	(in) The number of parameter values requested.
rgprmvd2	(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:

n_ulong nPnt	(in) point number
n_ushort nPrm	(in) parameter number
n_long nPrmOffset	(in) point parameter offset
PARvalue pupvValue*	(out) parameter value union
n_ushort nType	(out) value type
n_long fStatus	(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 * number of point parameters) < 4000
```

- For response packets when reading DT_INT2 data only:

```
(10 * number of points parameters) < 4000
```

- For response packets when reading DT_INT4 data only:

```
(12 * number of points parameters) < 4000
```

- For response packets when reading DT_REAL data only:

```
(12 * number of points parameters) < 4000
```

- For response packets when reading DT_DBLE data only:

```
(16 * number of points parameters) < 4000
```

- For response packets when reading DT_CHAR data only:

```
(9 * number of points parameters) + sum of string lengths of value character strings in bytes <
4000
```

- For response packets when reading DT_ENUM data only:

```
(13 * 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_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->dble);
                    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 344.

See also

“rhsc_param_value_puts_2” on page 313

“rhsc_param_value_put_bynames” on page 309

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
szHostname	(in) Name of server that the database resides on
cpntnd	(in) The number of point name resolutions requested
rgpntnd2	(in/out) Pointer to a series of POINT_NUMBER_DATA_2 structures (one array 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:

n_char szPntName*	(in) point name to resolve
n_ulong nPnt	(out) point number
n_long fStatus	(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 344.

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
server	(in) Name of server that the database resides on
num_points	(in) The number of points passed to rgetdat_xxxx in the getdat_xxxx_data argument
getdat_xxxx_data	(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:

int2 type	(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: RGETDAT_TYPE_INT2, RGETDAT_TYPE_INT4, RGETDAT_TYPE_REAL4, RGETDAT_TYPE_REAL8, RGETDAT_TYPE_STR, RGETDAT_TYPE_BITS
int2 file	(in) Absolute database file number to retrieve/store field.
int2 rec	(in) Record number in above file to retrieve/store field.
int2 word	(in) Word offset in above record to retrieve/store field.
int2 start_bit	(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.
int2 length	(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.
int2 flags	(in) Bit zero specifies the direction to read/write for circular files. (0 = newest record, 1 = oldest record)
union value	(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.
int2 status	(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:

short int2	Two byte signed integer.
long int4	Four byte signed integer.
float real4	Four byte IEEE floating point number.
double real8	Eight byte (double precision) IEEE floating point number.
char str*	Pointer to string to be stored. Note this string need not be NULL terminated, but must be of the length specified by length (see rgetdat_data structure description above).
unsigned short bits	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 344.

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
err	(in) The error number to lookup
texterr	(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
server	(in) Name of server that the database resides on
num_gethsts	(in) The number of points passed to rgethstpar_xxxx in the gethstpar_xxxx_data argument
gethstpar_xxxx_data	(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:

uint2 hist_type	(in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following: HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE
uint4 hist_start_date	(in) Start date of history to receive in Julian days (number of days since 1st January 1981).
ureal4 hist_start_time	(in) Start time of history to retrieve in seconds since midnight.
uint2 num_hist	(in) Number of history values per point to be retrieved.
uint2 num_points	(in) Number of points to be processed. MAXIMUM value allowed is 20.
uint2 point_type_nums*	(in) Array (of dimension num_points) containing the point type/numbers of the point history values to retrieve.
uint2 point_params*	(in) Array of (dimension num_points) containing the parameter numbers of the history values to retrive.
uchar archive path*	This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string.
real4 hist_values*	(out) Array (of dimension num_points * num_hist) to provide storage for the returned history values.
uint2 gethst_status	(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:

uint2 hist_type	(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
uint4 hist_offset	(in) Offset from latest history value in history intervals where offset=1 is the most recent history value).

uint2 num_hist	(in) Number of history values per point to be retrieved.
uint2 num_points	(in) Number of points to be processed. MAXIMUM value allowed is 20.
uint2 point_type_nums*	(in) Array (of dimension num_points) containing the point type/numbers of the point history values to retrieve.
uint2 point_params*	(in) Array of (dimension num_points) containing the parameter numbers of the history values to retrive.
uchar archive path*	This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string.
real4 hist_values*	(out) Array (of dimension num_points * num_hist) to provide storage for the returned history values.
uint2 gethst_status	(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 344.

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
server	(in) Name of server that the database resides on
num_points	(in) The number of points passed to rgetpnt in the getpnt_data argument
getpnt_data	(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:

char point_name*	(in) Pointer to a null terminated string containing the point name to be resolved into a point number.
uint2 point_type_num	(out) Return value of the point type/number for the point named above.
uint2 getpnt_status	(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 344.

See also

“rhsc_point_numbers_2” on page 319

rgetval_numb

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_numb
(
    char*               server,
    int                 num_points,
    rgetval_numb_data*  getval_numb_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_numb_data() As rgetval_numb_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
server	(in) Name of server that the database resides on
num_points	(in) The number of points passed to rgetval_xxxx in the getval_xxxx_data argument
getval_xxxx_data	(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_numb_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_numb_data is defined in *nif_types.h*. This structure and its members are defined as follows:

uint2 point_type_num	(in) Defines the point type/number to be accessed.
uint2 point_param	(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.
real4 param_value	(out) Value of the point parameter retrieved.
uint2 getval_status	(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:

uint2 point_type_num	(in) Defines the point type/number to be accessed.
uint2 point_param	(in) Defines the point parameter to be accessed (for example, description (DESC)). The definitions for all parameter types are located in the parameters file.
char param_value*	(out) NULL terminated string value of the point parameter. Note that this string can have a length of NIF_MAX_ASCII_PARAM_LEN +1 (for the termination), and that this amount of space must be allocated by the calling program.
uint2 param_len	(out) Useful length of above param_value retrieved (in bytes).
uint2 getval_status	(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:

uint2 point_type_num	(in) Defines the point type/number to be accessed.
uint2 point_param	(in) Defines the point parameter to be accessed (for example, 1 minute history, HST_1MIN). The definitions for all parameters are located in the parameters file.
uint2 hist_offset	(in) Offset from latest history value in history intervals to retrieve value, where hist_offset=1 is the most recent history value.
real4 param_value	(out) Value of the point parameter retrieved.
uint2 getval_status	(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 * number of points) + sum of all string value lengths in bytes <4000
```

Diagnostics

See "Diagnostics for Network API functions" on page 344.

See also

"rhsc_param_values_2" on page 316

"rhsc_param_value_puts_2" on page 313

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
server	(in) Name of server that the database resides on
point	(in) Name of point
param	(in) point parameter number
value	(out) Value of point parameter returned by function
length	(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 344.

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
server	(in) Name of server that the database resides on
num_requests	(in) The number of history requests
gethstpar_xxxx_data	(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:

uint2 hist_type	(in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following: HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE
uint4 hist_start_date	(in) Start date of history to receive in Julian days (number of days since 1st January 1981).
ureal4 hist_start_time	(in) Start time of history to retrieve in seconds since midnight.
uint2 num_hist	(in) Number of history values per point to be retrieved.
uint2 num_points	(in) Number of points to be processed. MAXIMUM value allowed is 20.
uint2 point_type_nums*	(in) Array (of dimension num_points) containing the point type/numbers of the point history values to retrieve.
uint2 point_params*	(in) Array of (dimension num_points) containing the parameter numbers of the history values to retrive.
uchar archive path*	This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string.

real4 hist_values*	(out) Array (of dimension num_points * num_hist) to provide storage for the returned history values.
uint2 gethst_status	(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:

uint2 hist_type	(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
uint4 hist_offset	(in) Offset from latest history value in history intervals where offset=1 is the most recent history value).
uint2 num_hist	(in) Number of history values per point to be retrieved.
uint2 num_points	(in) Number of points to be processed. MAXIMUM value allowed is 20.
uint2 point_type_nums*	(in) Array (of dimension num_points) containing the point type/numbers of the point history values to retrieve.
uint2 point_params*	(in) Array of (dimension num_points) containing the parameter numbers of the history values to retrieve.
uchar archive path*	This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string.
real4 hist_values*	(out) Array (of dimension num_points * num_hist) to provide storage for the returned history values.
uint2 gethst_status	(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 344.

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<= point number <= 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
szHostname	(in) Name of server that the database resides on
cprmvd	(in) The number of controls to parameters requested
rgprmvd	(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:

n_ushort nPnt	(in) point number
n_ushort nPrm	(in) parameter number
n_long nPrmOffset	(in) point parameter offset
PARvalue pupvValue*	(in) parameter value union
n_ushort nType	(in) value type
n_long fStatus	(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 344) 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 pntana1's SP value to 42.0 and change its DESC to say 'Funky description.'

```
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);

/* Set the point number, parameter number and offset for the point parameter. Allocate space,
```

```
assign a value, and set the type for pntana1.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 pntana1.DESC */
rgprmvd[1].nPnt = rgprmnd[1].nPnt;
rgprmvd[1].nPrm = rgprmnd[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 344.

See also

“rhsc_param_values” on page 334

“rhsc_param_value_put_bynames” on page 309

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<= point number <= 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
szHostname	(in) Name of server that the database resides on.
nPeriod	(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.
cprmvd	(in) The number of parameter values requested.
rgprmvd	(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:

n_ushort nPnt	(in) point number
n_ushort nPrm	(in) parameter number
n_long nPrmOffset	(in) point parameter offset
PARvalue pupvValue*	(out) parameter value union
n_ushort nType	(out) value type
n_long fStatus	(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->dble);
                    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 344.

See also

“rhsc_param_value_puts” on page 331

“rhsc_param_value_put_bynames” on page 309

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
szHostname	(in) Name of server on which the database resides
cprmnd	(in) The number of parameter name resolutions requested
rgprmnd	(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:

n_ushort nPnt	(in) point number
n_char szPrmName*	(in) parameter name to resolve
n_ushort nPrm	(out) parameter number returned
n_long fStatus	(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 * number of points) + sum of string lengths of point names in bytes < 4000
```

- For response packets:

```
(6 * 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 344.

See also

“rhsc_point_numbers” on page 339

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
szHostname	(in) Name of server that the database resides on
cpntnd	(in) The number of point name resolutions requested
rgpntnd	(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:

n_char szPntName*	(in) point name to resolve
n_ushort nPnt	(out) point number
n_long fStatus	(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 344.

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
server	(in) Name of server that the database resides on
point	(in) Name of point
param	(in) Point parameter number
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 344.

rputval_hist

Store history values.

This function's synopsis and description are identical to that of 'rputval_ascii.'

rputval_numb

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_numb
(
    char*    server,
    int    num_points,
    rputval_numb_data*  putval_numb_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
server	(in) Name of server that the database resides on
num_points	(in) The number of points passed to rputval_xxxx in the putval_xxxx_data argument
putval_xxxx_data	(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_numb_data structure is defined in *nif_types.h*. This structure and its members are defined as follows:

n_ushort point_type_num	(in) Defines the point type/number to be accessed.
n_ushort point_param	(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.
n_float param_value	(out) Value of point parameter to be stored.
n_short putval_status	(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:

n_ushort point_type_num	(in) Defines the point type/number to be accessed.
n_ushort point_param	(in) Defines the point parameter to be accessed. (for example, description (DESC)). The definitions for parameter type are located in the parameters file.
n_char param_value*	(in) ASCII string value of point parameter to be stored (Note this does not need to be null terminated).
uint2 param_len	(in) Length of above param_value to be stored (in bytes).
n_ushort putval_status	(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:

uint2 point_type_num	(in) Defines the point type/number to be accessed.
uint2 point_param	(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.
uint2 hist_offset	(in) Offset from latest history value in history intervals to store value. (Where hist_offset=1 is the most recent history value)
real4 param_value	(in) Value of point parameter to be stored.
uint2 putval_status	(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” on page 344.

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_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_WRONG_PROGRAM (0x8385)	The remote server being contacted has a NADS program number assignment other than that specified in the request.

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.

Batch Application Services

This section describes how to write applications for Experion using the Batch Application Services API.

Related topics

"About Batch Application Services" on page 348
"About the Batch Application Services development environment" on page 349
"Licensing Batch Application Services" on page 350
"Security considerations" on page 351
"BatchML Object Model" on page 352
"Function reference" on page 353
"Filtering" on page 369
"Diagnostics" on page 373

About Batch Application Services

Batch Application Services is an application programming interface (API) that application developers use to create, control, and interact with Experion Batch Manager Activities across Experion systems, including DSA-connected systems.

The API provides programmatic access to create, remove, command, monitor, and update top-level activities. Access is similar in scope to that of an operator using Station.

Batch Application Services supports the Experion security model requiring authentication and authorization of application users. That is, the application must run under an account that can be authenticated and has the necessary scope of responsibility (SOR) in Experion for the activities being accessed.

Experion server redundancy is supported by accessing the API through a client side component installed on the same node as the client application.

Batch Application Services is a licensable addition to Experion. For more information about licensing, contact your Honeywell representative.

Attention
In this document, the term *activities* represents both batch- and procedure-type activities.

Topology

"Figure 10: Batch Application Services topology" demonstrates the deployment of the Batch Application Services components. These components include:

Component	Description
Independent node	A node on which a Batch Application Services client is running. The Batch Application Services Client Component can be optionally installed here.
Client Component	The components installed by the Batch Application Services Client Component. This includes: • Native (C++) API • Redundancy Service
Client	A client application of the Batch Application Services API.

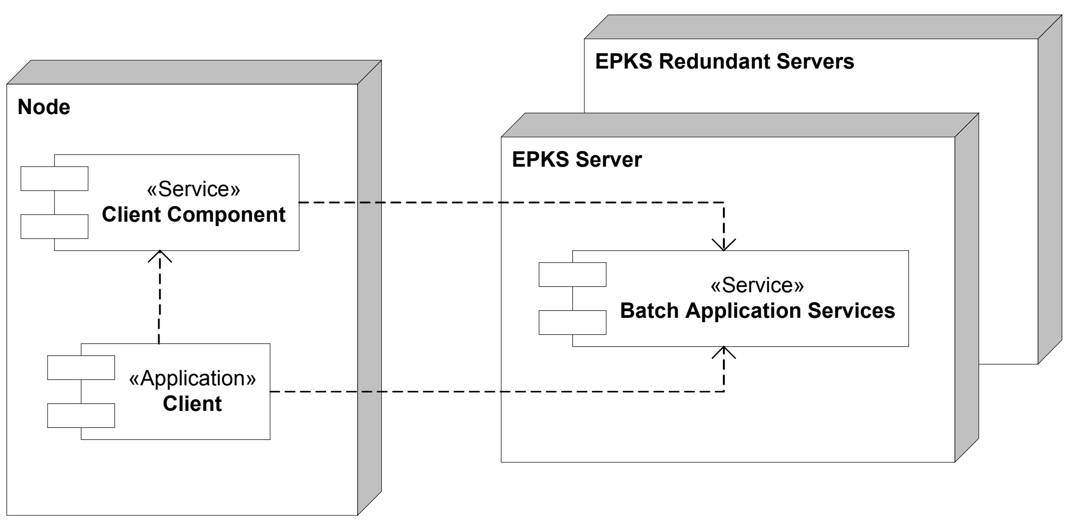


Figure 10: Batch Application Services topology

About the Batch Application Services development environment

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

Development against this API is intended to be performed using the Integrated Development Environment (IDE). As such, a good familiarity with this tool, as well as the C# or C++ programming languages, is advised.

Licensing Batch Application Services

Batch Application Services is a licensable addition to Experion and is available from server type nodes:

- Experion Server (ESV)
- Experion Server - TPS (ESV-T)
- Experion Application Server (EAS)

The **Server License Details** page in Station shows the license state (enabled or disabled) for Batch Application Services. When the license state is disabled, API calls to the server will generate an exception indicating that Batch Application Services is not licensed.

For more information about licensing, contact your Honeywell representative.

Security considerations

Client applications making requests on Batch Application Services functions must be authenticated and authorized before they are permitted to interact with the Experion system.

Authentication is performed using Windows infrastructure with the identity of the executing user account provided to the Experion server. This user account must either be defined as an *Operator* or *Automated System* account, or be a member of a group defined in the Experion Operators table.

Authorization is performed using the Windows user account's effective Scope of Responsibility (SOR) as defined in the Experion Operators table.

Attention

Traditional Operator accounts are not supported for use with Batch Application Services functions because they do not support windows authentication.

Automated System account

An *Automated System* account can only interact with the Experion system using an API or external interface, and should be used when the client application is acting on behalf of an automated system. This type of account does not provide user access to the system via Station.

An *Automated System* account is assigned the security level of *Program*, which cannot be modified.

For more information about *Automated System* accounts, see 'Adding an operator account' in the *Server and Client Configuration Guide*.

Console Stations

Batch Application Services API calls are not available when a client application attempts to make calls to a Console Station node.

Event logging

By default, Experion does not log as database events any changes made by an *Automated System* account using the Batch Application Services functions.

You can turn on event logging of Batch Application Services functions by selecting the *Log Application Services function changes by this account to the database as events* check box when setting up an *Automated System* account.

For more information about this setting, see 'Operator definition, General tab' in the *Server and Client Configuration Guide*.

Attention

Logging changes via Batch Application Services functions as events may flood the event log and adversely affect system performance.

BatchML Object Model

Batch Markup Language (BatchML) is a referenced XML schema developed by *WBF - The Organization for Production Technology* (previously known as the *World Batch Forum*), and is used to exchange information about recipes, equipment, and batch lists. Copies of the schema are available from WBF at “http://www.wbf.org”.

BatchML is based on the data models and attributes defined in the ANSI/ISA 88.00.02 Batch Control standard Part 2. Contact ISA (The Instrumentation, Systems, and Automation Society) for copies of the standard. For more information about the standard, go to “http://www.isa.org”.

Batch Application Services's object model is based on BatchML version V0401.

See “BatchML Object Model class diagrams” on page 463 for the BatchML Object Model class diagrams that Batch Application Services uses.

Function reference

This section describes the methods in Batch Application Services.

Related topics

"IsPrimary" on page 353
"GetActivityList" on page 353
"GetActivityEntityList" on page 356
"GetActivityEntityMetadata" on page 358
"CreateActivity" on page 361
"SetPointParameterValues" on page 363
"GetPointParameterValues" on page 365

IsPrimary

Description

The *IsPrimary* method retrieves the current state of the server. It returns *true* if the target server is primary, and *false* if it is not.

IsPrimary is a blocking function.

> **Attention**
> When using the Redundancy Service and making a call to *IsPrimary()* using the redundant pairs base name, the *IsPrimary()* API will always return *true*. This is because the call will always be directed to the primary server.

Syntax C#

Namespace: Varies depending on how the client consumes the service.

```
[OperationContract]
bool IsPrimary()
```

Syntax C++

Namespace: *Honeywell::Client::Application::Services::API*

```
bool IsPrimary()
```

Exceptions

See "Diagnostics" on page 373.

GetActivityList

Description

The *GetActivityList* method retrieves all of the activities from the server that are within the caller's SOR, applies the filters defined in *activityFilterList*, and returns the matching activities in *activityList*.

activityList is a *BatchInformationType* object and acts as a container for the activities. "Figure 11: BatchInformationType structure for GetActivityList" describes a simplified structure of *BatchInformationType* and its components for *activityList*. For more information about *BatchInformationType*, see "BatchML Object Model class diagrams" on page 463.

GetActivityList is a blocking function.

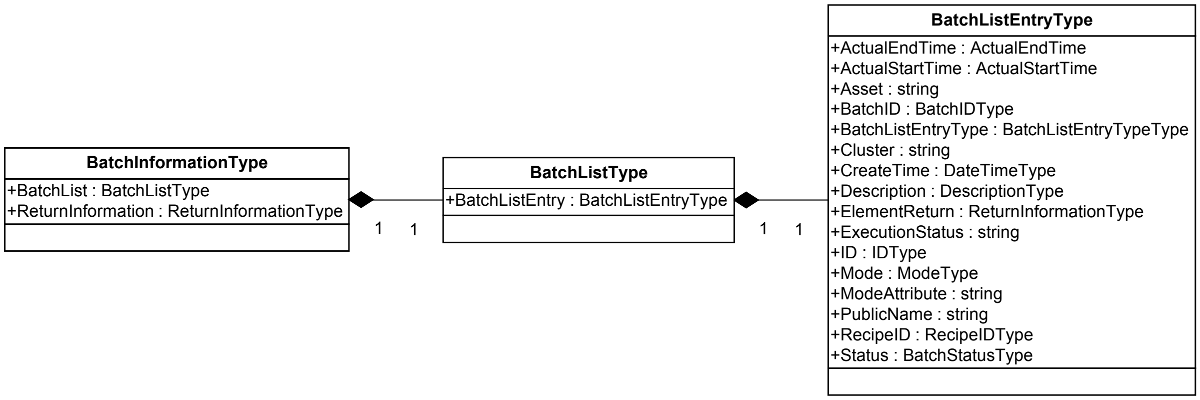


Figure 11: BatchInformationType structure for GetActivityList

DataMember	Description
BatchList	List of activities.
ReturnInformation	Status of the call.
BatchList.BatchListEntry	An activity.
BatchListEntry.ActualEndTime	Time the activity ended. Example: *2012-12-18T17:00:00.000+11:00*
BatchListEntry.ActualStartTime	Time the activity started. Example: *2012-12-18T09:00:00.000+11:00*
BatchListEntry.Asset	Asset to which the activity belongs. Example: *MixerA*
BatchListEntry.BatchID	Value of the batch ID. Example: *Batch B000123*
BatchListEntry.BatchListEntryType	Type of the activity (*Batch* or *Procedure*). Example: *Batch*
BatchListEntry.Cluster	Server base name of the activity. Example: *Server1*
BatchListEntry.CreateTime	Time the activity was created. Example: *2010-12-18T08:55:00.000+11:00*
BatchListEntry.Description	Description of the activity. Example: *Super strong resin*
BatchListEntry.ElementReturn	Detailed status of call.
BatchListEntry.ExecutionStatus	Executing status of the activity. Example: *OK*
BatchListEntry.ID	Tag name of the activity. Example: *SvrB:$Activity000123*
BatchListEntry.Mode	Mode of the activity. Example: *AUTO*
BatchListEntry.ModeAttribute	Mode attribute of the activity. Example: *PROGRAM*

DataMember	Description
BatchListEntry.PublicName	Public name of the activity. Example: *Red paint recipe*
BatchListEntry.RecipeID	Tag name of the activity entity. Example: *ServerB:RCM_1*
BatchListEntry.Status	State of the activity. Example: *STOPPED*

Syntax C#

Namespace: Varies depending on how the client consumes the service.

```
[OperationContract]
int GetActivityList(
    ActivityFilterList activityFilterList,
    out BatchInformationType activityList
    )
```

Parameters	Description
activityFilterList [in]	Type: *Honeywell.Client.Application.Services.AppServicesRef.BatchInformationType.ActivityFilterList* Filter definitions: see “Filtering” on page 369.
activityList [out]	Type: *Honeywell.Client.Application.Services.AppServices.AppServicesRef.BatchInformationType* BatchML structure with *BatchListEntry* types populated after call.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Syntax C++

Namespace: *Honeywell::Client::Application::Services::API*

```
int GetActivityList(
    ActivityFilterList* activityFilterList,
    BatchInformationType*& activityList
    )
```

Parameters	Description
activityFilterList [in]	Type: *Honeywell ::Client::Application::Services::XMLTranslate::DataContract::FilterN.ActivityFilterList* Filter definitions: see “Filtering” on page 369.
activityList [out]	Type: *Honeywell::Client::Application::Services::XMLTranslate::BatchInformation.BatchInformationType* BatchML structure with MasterRecipe types populated after call.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Attention

For the C++ interface, the caller is responsible for releasing the memory allocated to the out parameter; this is done by calling *delete* on this parameter. All memory associated with the data members in this parameter will be automatically released.

Exceptions

See “Diagnostics” on page 373.

Examples

See the *Returning only one activity* section in 'Getting more information about an activity entity' for a usage example of *GetActivityList*. This contains both C# and C++ code.

GetActivityEntityList

Description

The *GetActivityEntityList* method retrieves all of the activity entities from the server that are within the caller's SOR, applies the filters defined in *activityEntityFilterList*, and returns the matching activity entities in *activityEntityList*.

activityEntityList is a *BatchInformationType* object and acts as a container for the activity entities. “Figure 12: BatchInformationType structure for GetActivityEntityList” describes a simplified structure of *BatchInformationType* and its components for *activityEntityList*. For more information about *BatchInformationType*, see 'BatchML Object Model class diagrams.'

GetActivityEntityList is a blocking function.

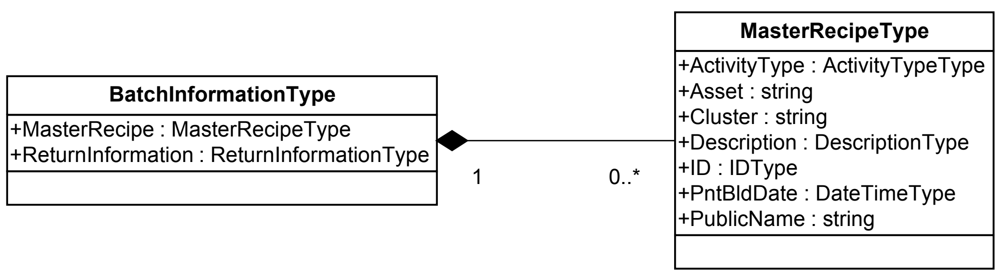


Figure 12: BatchInformationType structure for GetActivityEntityList

DataMember	Description
MasterRecipe[*]	List of activity entities.
ReturnInformation	Status of the call.
MasterRecipe.ID	Tag name of the activity entity. Example: *Server1:RCM_MakeResin*
MasterRecipe.Description	Description of the activity entity. Example: *Super strong resin*
MasterRecipe.Cluster	Server base name of the activity entity. Example: *Server1*
MasterRecipe.Asset	Asset to which the activity entity belongs. Example: *Asset01*

DataMember	Description
MasterRecipe.ActivityType	Type of the activity entity. Example: *Batch*
MasterRecipe.PublicName	Public name of the activity entity. Example: *Resin recipe*
MasterRecipe.PntBldDate	The point build date of the activity entity. Example: *2011-12-16T09:10:00.000+11:00*

Syntax C#

Namespace: Varies depending on how the client consumes the service.

```
[OperationContract]
int GetActivityEntityList(
    ActivityEntityFilterList activityEntityFilterList,
    out BatchInformationType activityEntityList
    )
```

Parameters	Description
activityEntityFilterList [in]	Type: *Honeywell.Server.Application.Services.XMLTranslate.DataContract.Filter.ActivityEntityFilterList* Filter definitions: see “Filtering” on page 369.
activityEntityList [out]	Type: *Honeywell.Server.Application.Services.XMLTranslate.BatchInformation.BatchInformationType* BatchML structure with *MasterRecipe* types populated after call.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Syntax C++

Namespace: *Honeywell::Client::Application::Services::API*

```
int GetActivityEntityList(
    ActivityEntityFilterList* activityEntityFilterList,
    BatchInformationType*& actEntityList
    )
```

Parameters	Description
activityEntityFilterList [in]	Type: *Honeywell ::Client::Application::Services::XMLTranslate::DataContract::FilterN.ActivityEntityFilterList* Filter definitions: see “Filtering” on page 369.
activityEntityList [out]	Type: *Honeywell::Client::Application::Services::XMLTranslate::BatchInformation.BatchInformationType* BatchML structure with *MasterRecipe* types populated after call.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Attention

For the C++ interface, the caller is responsible for releasing the memory allocated to the out parameter; this is done by calling `delete` on this parameter. All memory associated with the data members in this parameter will be automatically released.

Exceptions

See "Diagnostics" on page 373.

Examples

See 'Getting a list of activity entities' for a usage example of *GetActivityEntityList*. This contains both C# and C++ code.

GetActivityEntityMetadata

Description

The *GetActivityEntityMetadata* method retrieves the metadata associated with the activity or activity entity specified in *activityEntityMetadata*.

On input, *activityEntityMetadata* should contain one *MasterRecipe* with an ID that is the same as the tag name of the activity or activity entity.

On output, the only *MasterRecipe* in *activityEntityMetadata* is populated with metadata of the activity or activity entity.

If the activity or activity entity is a *class based recipe* (CBR), the *EnumerationSet* field in *activityEntityMetadata* contains the information about the units used in the CBR.

"Figure 13: BatchInformationType structure for GetActivityEntityMetadata" describes a simplified structure of *BatchInformationType* and its components for *GetActivityEntityMetadata*. For more information about *BatchInformationType*, see 'BatchML Object Model class diagrams.'

GetActivityEntityMetadata is a blocking function.

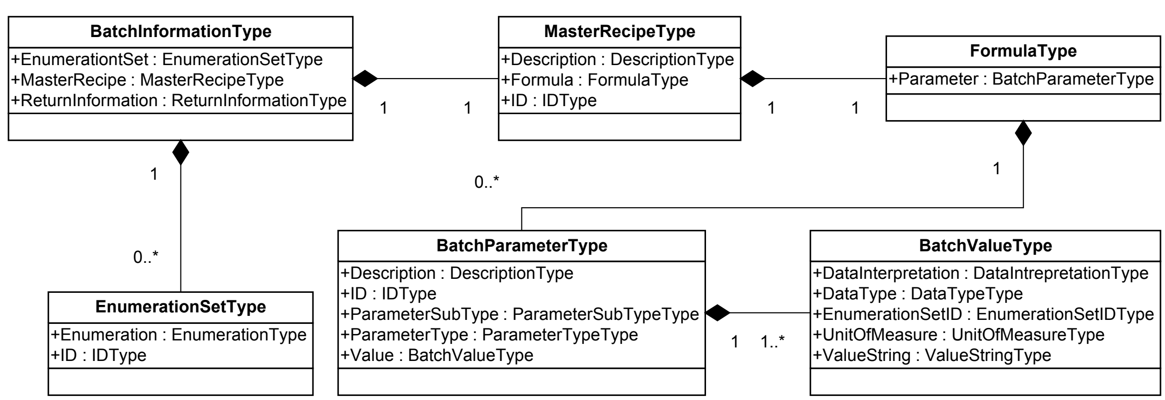


Figure 13: BatchInformationType structure for GetActivityEntityMetadata

DataMember	Description
MasterRecipe	The activity entity.
EnumerationSet	Contains unit information. Used only in class based recipes.
ReturnInformation	Status of the call.

DataMember	Description
MasterRecipe.ID	Tag name of the activity entity. Example: *Server1:RCM_MakeResin*
MasterRecipe.Description	Description of the activity entity. Example: *Super strong resin*
MasterRecipe.Formula.Parameter[*]	List of parameters in the activity entity.
MasterRecipe.Formula.Parameter.ID	Tag name of the parameter. Example: *BLOCK.PARAM1*
MasterRecipe.Formula.Parameter. Description	Description of the parameter. Example: *Boiler Temperature*
MasterRecipe.Formula.Parameter. ParameterType	Type of the parameter. • *ProcessInput* • *ProcessOutput* • *Other* Use *Other* when the parameter type is *Header*. Set the *OtherValue* field to *Header*. Example: *ProcessInput*
MasterRecipe.Formula.Parameter. ParameterSubType	Indicates if the parameter can be edited. • *Read Only* • Editable Example: *Read Only*
MasterRecipe.Formula.Parameter. Value.Primary	Indicates if the unit is a primary unit. Valid only for class-based recipes. • *true* • false Example: *true*
MasterRecipe.Formula.Parameter. Value[*]	List of values for the parameter
MasterRecipe.Formula.Parameter. Value.ValueString	Actual value definition. Example: *BLOCK.PARAM1MINVALUE*
MasterRecipe.Formula.Parameter. Value.Description	Indicates whether the value is the default, minimum, or maximum value for the parameter. • *Default* • Min • *Max* Example: *Default*

DataMember	Description
MasterRecipe.Formula.Parameter.Value.DataInterpretation	Indicates how to interpret the value for the parameter. • Constant • Other Use *Other* when the *ValueString* is another point parameter. Set the *OtherValue* field to *Header*. • When the point parameter is from an activity, set the *OtherValue* field to *ActivityReference*. • When the point parameter is from an activity entity, set the *OtherValue* field to *ActivityEntityReference*. Example: *Constant*
MasterRecipe.Formula.Parameter.Value.DataType	Type of the value for the parameter. Example: *string*
MasterRecipe.Formula.Parameter.Value.UnitOfMeasure	Type of the value for the parameter. Example: *Fahrenheit*
MasterRecipe.Formula.Parameter.Value.EnumerationSetID	Specifies the ID of the *EnumerationSet* that contains the unit information for this value. Valid only for class-based recipes. Example: *Unit2*

Syntax C#

Namespace: Varies depending on how the client consumes the service.

```
[OperationContract]
int GetActivityEntityMetadata(
    ref BatchInformationType activityEntityMetadata
    )
```

Parameters	Description
activityEntityMetadata [in/out]	Type: *Honeywell.Server.Application.Services.XMLTranslate.BatchInformation.BatchInformationType* On input, contains the tag name of the activity or activity entity to retrieve metadata. On output, contains the metadata of the activity or activity entity.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Syntax C++

Namespace: *Honeywell::Client::Application::Services::API*

```
int GetActivityEntityMetadata(
    BatchInformationType*& activityEntityMetadata
    )
```

Parameters	Description
activityEntityMetadata [in/out]	Type: *Honeywell::Client::Application::Services::XMLTranslate::BatchInformation.BatchInformationType* On input, contains the tag name of the activity or activity entity to retrieve metadata. On output, contains the metadata of the activity or activity entity.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Attention
For the C++ interface, the caller is responsible for releasing the memory allocated to the out parameter; this is done by calling *delete* on this parameter. All memory associated with the data members in this parameter will be automatically released.

Exceptions

See “Diagnostics” on page 373.

Examples

See 'Getting more information about an activity entity' for a usage example of *GetActivityEntityMetadata*. This contains both C# and C++ code.

CreateActivity

Description

The *CreateActivity* method accepts a BatchML *BatchInformation* object containing a *ControlRecipe* object.

The *ControlRecipe* object is required to have:

- *ControlRecipe.ID* set to the activity entity's tag name.
- *ControlRecipe.BatchID* set to the activity's batch ID. The Batch ID is only mandatory when creating an activity from a batch type activity entity.

This information is then used to create an activity from the activity entity and assign it the supplied batch ID.

“Figure 14: BatchInformationType structure for CreateActivity” and “Figure 15: BatchListType structure for CreateActivity” describe simplified structures of *BatchInformationType* and its components for *CreateActivity*. For more information about *BatchInformationType*, see 'BatchML Object Model class diagrams.'

CreateActivity is a blocking function.

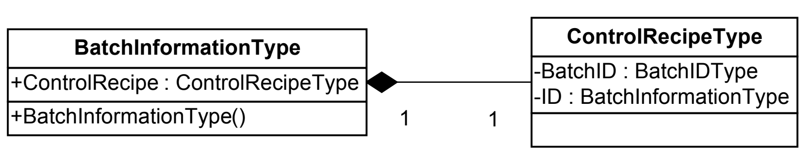


Figure 14: BatchInformationType structure for CreateActivity

DataMember	Description
ControlRecipe	The activity to be created.

DataMember	Description
ControlRecipe.BatchID	Value of the batch ID. Example: *Batch XYZ*
ControlRecipe.ID	Tag name of the activity entity. Example: *SvrB:RCM_1*

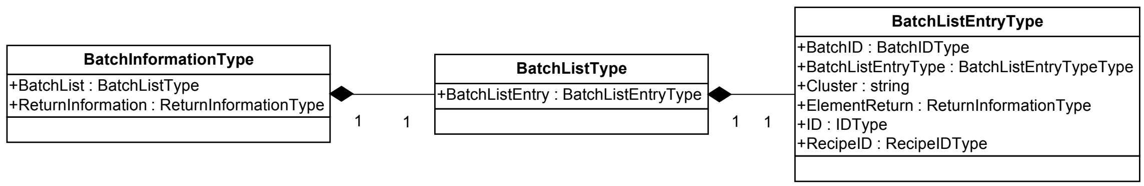


Figure 15: BatchListType structure for CreateActivity

DataMember	Description
BatchList	An activity list, containing one activity.
ReturnInformation	Status of the call.
BatchListEntry	The newly created activity.
BatchListEntry.BatchID	Value of the batch ID. Example: *Batch XYZ*
BatchListEntry.BatchListEntryType	Type of the activity (*Batch* or *Procedure*). Example: *Batch*
BatchListEntry.Cluster	Server base name of the activity. Example: *Server1*
BatchListEntry.ElementReturn	Detailed status of call.
BatchListEntry.ID	Tag name of the activity. Example: *SvrB:$Activity000123*
BatchListEntry.RecipeID	Tag name of the activity entity. Example: *ServerB:RCM_1*

Syntax C#

Namespace: Varies depending on how the client consumes the service.

```
[OperationContract]int CreateActivity(
    BatchML.BatchInformationType ActivityTemplate;
    out BatchML.BatchInformationType activityInstance
    )
```

Parameters	Description
activityTemplate [in]	Type: *Honeywell.Client.Application.ServicesRef.Ref.BatchInformationType* BatchML structure containing information required to create an activity.
activityInstance [out]	Type: *Honeywell.Client.Application.ServicesRef.Ref.BatchInformationType* BatchML structure containing information about the newly created activity.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Syntax C++

Namespace: *Honeywell::Client::Application::Services::API*

```
int CreateActivity(
    NativeBatchML::BatchInformationType* activityTemplate,
    NativeBatchML::BatchInformationType*& activityInstance
    )
```

Parameters	Description
activityTemplate [in]	Type: *NativeBatchML::BatchInformationType* BatchML structure containing information required to create an activity.
activityInstance [out]	Type: *NativeBatchML::BatchInformationType* BatchML structure containing information about the newly created activity.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Attention

For the C++ interface, the caller is responsible for releasing the memory allocated to the out parameter; this is done by calling *delete* on this parameter. All memory associated with the data members in this parameter will be automatically released.

Exceptions

See “Diagnostics” on page 373.

Examples

See 'Creating an activity' for a usage example of *CreatetActivity*. This contains both C# and C++ code.

SetPointParameterValues

Description

The *SetPointParameterValues* method sets point parameter values on an Experion system.

This operation accepts a BatchML extension *PointParamType* object containing a *PointList* object. This object will contain one or more points, with each point containing one or more parameters. Each parameter will be populated with the parameter value.

A successful call will result in each parameter value being set on the Experion system.

“Figure 16: BatchInformationType structure for SetPointParameterValues” describes a simplified structure of *BatchInformationType* and its components for *SetPointParameterValues*. For more information about *BatchInformationType*, see 'BatchML Object Model class diagrams.'

SetPointParameterValues is a blocking function.

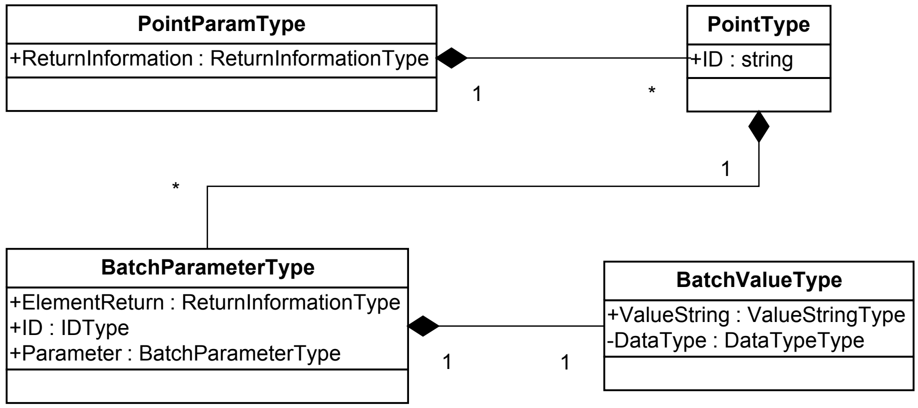


Figure 16: BatchInformationType structure for SetPointParameterValues

DataMember	Description
PointParamType.PointList	List of points.
PointParamType.ReturnInformation	Status of the call.
PointType.Parameter	List of parameters to have value read.
PointType.ID	Point name. Example: *Sinewave*
BatchParameterType.ElementReturn	Detailed status of the call
BatchParameterType.ID	Parameter name.
BatchValueType.DataType	Experion type of the parameter. See “Supported data types”. Example: *INT_16*
BatchValueType.ValueString	The parameter value. Null if a reference has been passed to the parameter. Example: *10.5*

Syntax C#

Namespace: Varies depending on how the client consumes the service.

```
[OperationContract]
int SetPointParameterValues(ref BatchML.PointParamType batches)
```

Parameters	Description
batches [in/out]	Type: *<namespace>.PointParamType* BatchML extension structure containing information about a list of points and their parameters.

Return value type	Notes
int	Error code: see “Error codes” on page 373.

Syntax C++

Namespace: *Honeywell::Client::Application::Services::API*

```
int AppServicesAPI::SetPointParameterValues (
    NativeBatchML::PointParamType*& batches
    )
```

Parameters	Description
batches [in/out]	Type: *NativeBatchML::PointParamType* BatchML extension structure containing information about a list of points and their parameters.

Return value type	Notes
int	Error code: see "Error codes" on page 373.

Attention

For the C++ interface, the caller is responsible for releasing the memory allocated to the out parameter; this is done by calling *delete* on this parameter. All memory associated with the data members in this parameter will be automatically released.

Supported data types

Under normal usage, *DataType* is ignored and the *ValueString* converted to an Experion *PARValue* according to the internal data type of the point parameter. The exception to this is when explicitly writing an enumeration type point parameter by its ordinal value, in which case *DataType* should be set to "short."

Exceptions

See "Diagnostics" on page 373.

Examples

See 'Configuring an activity before starting it' for a usage example of *SetPointParameterValues*. This contains both C# and C++ code.

GetPointParameterValues

Description

If your system uses *dynamic scanning*, GetPointParameterValues calls from Batch Application Services do not trigger dynamic scanning.

The *GetPointParameterValues* method retrieves point parameter values from an Experion system.

This operation accepts a BatchML extension *PointParamType* object containing a *PointList* object and a subscription period. The *PointList* object will contain one or more points, with each point containing one or more parameters. The subscription period argument is used to specify the rate, if any, at which these point parameters should be subscribed.

A successful call will result in the *PointParam* object being returned with the point parameter values.

"Figure 17: BatchInformationType structure for GetPointParameterValues" describes a simplified structure of *BatchInformationType* and its components for *GetPointParameterValues*. For more information about *BatchInformationType*, see 'BatchML Object Model class diagrams.'

GetPointParameterValues is a blocking function.

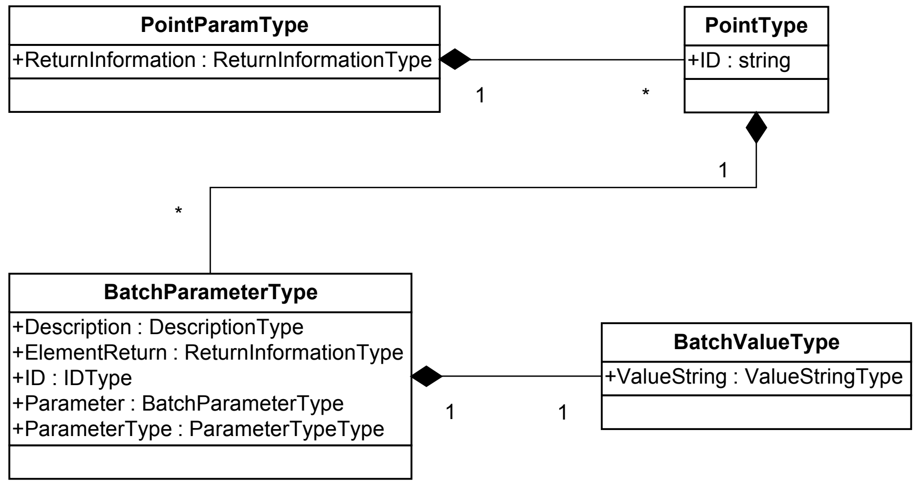


Figure 17: BatchInformationType structure for GetPointParameterValues

DataMember	Description
PointParamType.PointList	List of points.
PointParamType.ReturnInformation	Status of the call.
PointType.Parameter	List of parameters to have value read.
PointType.ID	Point name. Example: *Sinewave*
BatchParameterType.ElementReturn	Detailed status of the call
BatchParameterType.ID	Parameter name.
BatchParameterType.ParameterType	Experion type of the parameter. See “Supported data types”. Example: *INT_16*
BatchParameterType.ValueString	The parameter value. Null if a reference has been passed to the parameter. Example: *10.5*

Syntax C#

Namespace: Varies depending on how the client consumes the service.

```
[OperationContract]
int GetPointParameterValues(
    ref BatchML.PointParamType batches,
    int period)
```

Parameters	Description
batches [in/out]	Type: *<namespace>.PointParamType* BatchML extension structure containing information about a list of points and their parameters.
Period [in]	Subscription period in milliseconds for the point parameters. Use the constant *ESubscriptionPeriod.CACHE_READ* 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 *ESubscriptionPeriod.DEVICE_READ* if you want to force Experion to poll the controller again.

Return value type	Notes
int	Error code: see "Error codes" on page 373.

Syntax C++

Namespace: *Honeywell::Client::Application::Services::API*

```
int AppServicesAPI::GetPointParameterValues(
    NativeBatchML::PointParamType*& batches,
    int period)
```

Parameters	Description
batches [in/out]	Type: *NativeBatchML::PointParamType* BatchML extension structure containing information about a list of points and their parameters.
Period [in]	Subscription period in milliseconds for the point parameters. The subscription period will not be applied to standard point types. Use the constant *SUBSCRIPTION_CACHE_READ* 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 *SUBSCRIPTION_DEVICE_READ* if you want to force Experion to poll the controller again.

Return value type	Notes
int	Error code: see "Error codes" on page 373.

Attention
For the C++ interface, the caller is responsible for releasing the memory allocated to the out parameter; this is done by calling *delete* on this parameter. All memory associated with the data members in this parameter will be automatically released.

Return data type

Application Services Batch supports only a subset of Experion *PARValue* data types.

The following table maps the supported Experion data types to the equivalent BatchML DataType fields.

Experion Data Types	BatchML DataType field
DT_ENUM	*Enumeration*
DT_DSTADDR *DT_SRCADDR* *DT_CHAR*	*string*
DT_INT2	*short*
DT_INT4	*integer*
DT_INT8	*long*
DT_REAL *DT_HST*	*float*
DT_DBLE	*double*
DT_DATE_TIME *DT_TIME*	*Other* *OtherType: DateTime*

Requesting any other data type will result in the error 'Application Services - Parameter data type not supported' being returned to the client.

Subscription

A point parameter subscription period must be specified via the period argument. This value indicates how often the cached value of a point parameter will be updated.

This information is most relevant in a polling scenario. Care should be taken that the value supplied for the period argument matches the rate at which `GetPointParameterValues` is called. That is, if `GetPointParameterValues` is polled in a loop with a 5-second delay, then use a value of 5,000 (milliseconds) for the period argument.

No action needs to be taken to unsubscribe the point parameters since the Experion server will remove them from subscription after a period of inactivity.

Exceptions

See "Diagnostics" on page 373.

Examples

See the *Retrieving values of point parameters* section in 'Getting more information about an activity' for a usage example of `GetPointParameterValues`. This contains both C# and C++ code.

Filtering

Use the *GetActivityList* and *GetActivityEntityList* operations to apply filtering. The filtering work will be performed on the server and can result in smaller lists being returned and better system performance.

Filtering functionality is used by constructing and combining the filter objects documented below and passing them as an argument to either the *GetActivityList* or *GetActivityEntityList* operations.

See 'Filtering activity lists and activity entity lists' for more information.

Related topics

"Filtering class diagrams" on page 369
"UTC offset for DateTime filtering" on page 372

Filtering class diagrams

This section contains the class definitions used for filtering in Batch Application Services.

ActivityEntityFilterList

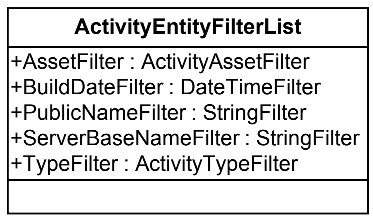


Figure 18: ActivityEntityFilterList

ActivityFilterList

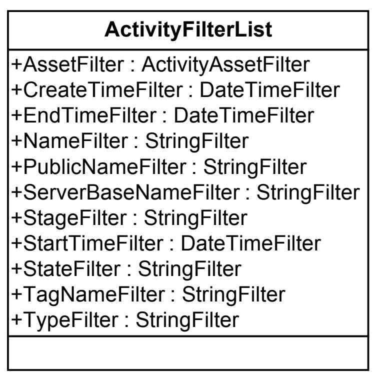


Figure 19: ActivityFilterList

ActivityAssetFilter

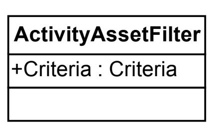


Figure 20: ActivityAssetFilter

ActivityTypeFilter

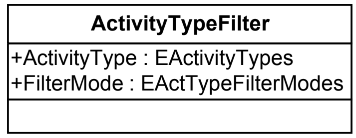


Figure 21: ActivityTypeFilter

CriteriaAnd

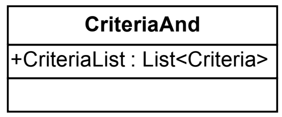


Figure 22: CriteriaAnd

CriteriaAsset

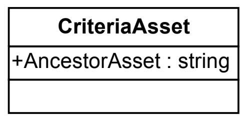


Figure 23: CriteriaAsset

CriteriaDateTime

`CriteriaDateTime` has two class diagrams: one for C# and one for C++.

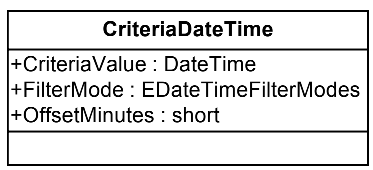


Figure 24: CriteriaDateTime - C#

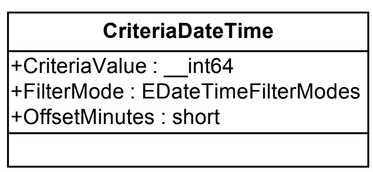


Figure 25: CriteriaDateTime - C++

CriteriaOr

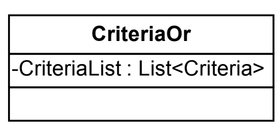


Figure 26: CriteriaOr

CriteriaString

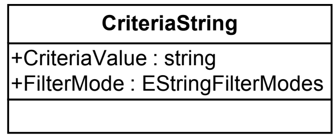


Figure 27: CriteriaString

DateTimeFilter

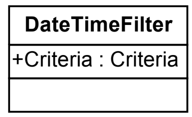


Figure 28: DateTimeFilter

EActivityTypes

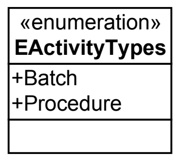


Figure 29: EActivityTypes

EActTypeFilterModes

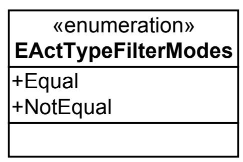


Figure 30: EActTypeFilterModes

EDateTimeFilterModes

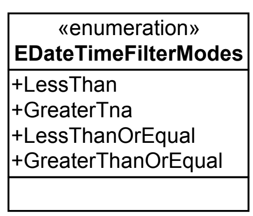


Figure 31: EDateTimeFilterModes

EStringFilterModes

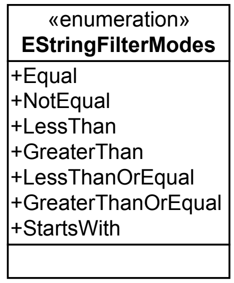


Figure 32: EStringFilterModes

StringFilter

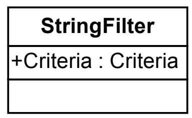


Figure 33: StringFilter

UTC offset for DateTime filtering

When specifying the value of a *CriteriaDateTime*, you specify the *DateTime* value in the local time of the server. Additionally, you must also identify the time zone of the server by specifying the Coordinated Universal Time (UTC) offset (in minutes).

> **Attention**
>
> *DateTime* uses the local time of the specified UTC offset as the basis for filtering. If you specify a UTC offset that is different to that of the server's time zone, the filtered results will not be accurate.

When filtering activities on *DateTime* values where the client and server are in different time zones, care must be taken to provide the correct UTC offset. Differences in UTC offsets could be due to geographically different locations or changes to or from daylight saving time (DST). If you specify a UTC offset other than that of the server's time zone, the filtering result will be shifted by the difference in the UTC offsets. For example, if filtering on *DateTime* values created during DST and a UTC offset of non-DST is provided, filtering will be shifted by one hour (assuming the DST shift is one hour).

Diagnostics

Batch Application Services uses two forms of error handling to inform the client application of an error or warning. These return codes could be from various Experion subsystems.

Related topics

"Error codes" on page 373
"WCF exceptions" on page 373

Error codes

While running or debugging your application, you may encounter error codes. These can originate not just from the Batch Application Services API, but from many other sources. For example, you may see an error code that the API has generated, or an SOR-related error code from Experion.

Error codes can be returned in two forms: as an integer value directly returned by the called API, or as a hexadecimal string contained in either a *ReturnInformation* or *ElementReturn* object.

The following error codes may be returned to the Batch Application Services API.

Error code	Description
SUCCESS (0x0)	Operation has returned without error.
INTERNAL_ERROR (0xC8CA)	An internal error occurred during operation processing.
INVALID_ARG (0xC8C9)	An input parameter is invalid.
INVALID_DATATYPE (0xC8CB)	Parameter data type not supported.
JOURNAL_ERROR (0xC8D1)	Adding parameter write journaling error.
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_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_GDA_WARNING (0x8151)	The GDA subsystem has reported a warning.
MISSING_ELEMENT (0xC8C8)	XML input is missing a required element.
OPC_QUALITY_NOT_GOOD (0xC8CF)	The quality of the data is not good.
PARTIAL_FUNC_FAIL (0xC8C7)	Operation returned at least one element in error.
PERMISSION_DENIED (0xC8CC)	Point is not within scope of responsibility (SOR).
TIMEOUT (0xC8CD)	Operation exceeded time limit.

WCF exceptions

Exceptions are thrown if there is an issue encountered in the transport layer or when the service throws a *FaultException*.

The following table maps C++ exceptions to their equivalent WCF C# exceptions. These C++ exceptions maintain the same inheritance hierarchy to their C# equivalents.

C# exception	C++ equivalent	Description
Exception	*NativeException*	Represents errors that occur during application execution.
TimeoutException	*NativeTimeoutException*	The exception that is thrown when the time allotted for a process or operation has expired.
CommunicationException	*NativeCommunicationException*	Represents a communication error in either the service or client application.
FaultException	*NativeFaultException*	Represents a SOAP fault. These faults contain additional error information, such as 'Server is not primary' and 'Operation is not licensed.'

For more information about these exception types, refer to the *Handling Errors* section of the MSDN article titled *WCF Client Overview* at http://msdn.microsoft.com/en-us/library/ms735103.aspx.

Batch Application Services tutorials

This section provides tutorials that show you how to write applications for Experion using the Batch Application Services API.

The tutorials provide examples in both C# and C++ programming languages.

> **Attention**
> Before you begin the tutorials, review the section “About Batch Application Services” on page 348 to learn about the topology of Batch Application Services.

Related topics

“Prerequisites for a C# project” on page 376
“Prerequisites for a C++ project” on page 383
“Simple tutorial” on page 394
“Intermediate tutorial” on page 412
“Error scenarios and exception handling” on page 435
“Connecting to multiple servers” on page 438
“Managing redundancy” on page 439
“Installing the Batch Application Services Client Component” on page 440
“Configuring the Batch Application Services Client Component” on page 443
“Configuring your client application to redirect messages to the redundancy service” on page 444
“Using client proxies to communicate with redundant servers” on page 445
“Using Class Based Recipes” on page 447

Prerequisites for a C# project

You will learn how to create a new C# project, consume the WCF service, and adjust *app.config* to support large list operations.

Disclaimer

Before commencing this tutorial the Batch Application Services Client Component should be installed. (See “Installing the Batch Application Services Client Component” on page 440.) If the Batch Application Services Client Component is not installed, then the directories referred to won't exist and runtime linkage errors may occur.

Related topics

“Creating a new C# project” on page 376
“Consuming the WCF Service” on page 377
“Adjusting app.config to support large list operations” on page 381

Creating a new C# project

Attention
This tutorial uses a *Console Application* project. However, you can use any C# project type.

To create a new C# Console Application project

1. Start Visual Studio.
2. From the File menu, click **New** > **Project**.
   The **New Project** window appears.
3. From the *Installed Templates* list, click **Visual C#** > **Windows**.
4. From the middle pane, click **Console Application**.
5. Name your project. Type `HelloBatchApplicationServicesCS` in the **Name** box.
6. Specify where to save the project. Type `c:\Tutorial` in the **Location** box.
   If you prefer, you can specify an alternate location to save the project.

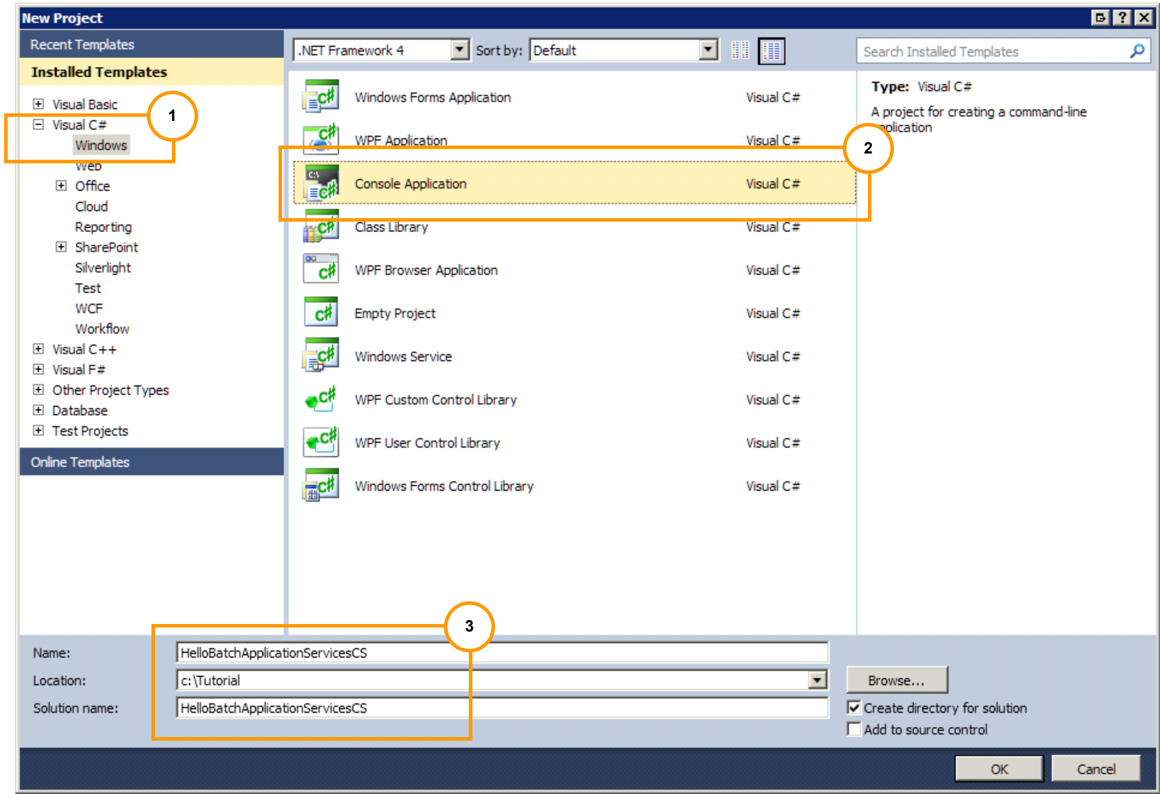


Figure 34: Creating a new C# project

Item	Description
1	List of installed templates, grouped by programming language.
2	List of available templates for the selected group.
3	File name, Location, and Solution name information.

7 Click **OK**.
Visual Studio creates the project.

Consuming the WCF Service

Now that you have created your project, you can configure it to consume the Windows Communication Foundation (WCF) Service. This step exposes an application's functionality as a service so that other applications can access (consume) it.

To consume the WCF Service

1 From the *Solution Explorer* list, right-click the **HelloBatchApplicationServicesCS** project, and then click **Add Service Reference**.
The **Add Service Reference** window appears.

Tip
If the **Add Service Reference** menu option is not visible, make sure that the project is targeting .Net version 3.5 or later.

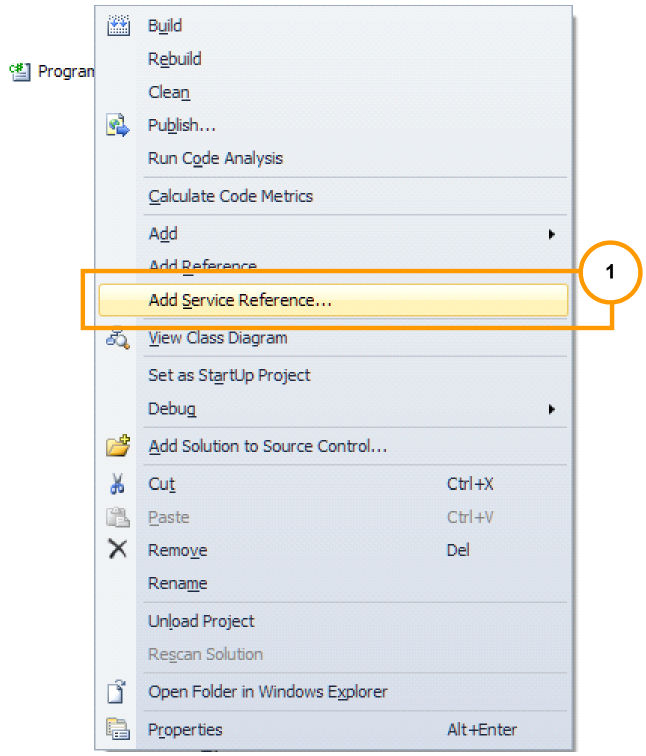


Figure 35: Add Service Reference menu

Item	Description
1	Add Service Reference menu item.

2 Type the following address in the **Address** box:

```
http://<servername>/Services/AppServices/Activity/mex
```

where `<server name>` is the name of the server to which you will connect. The example in “Figure 36: Add Service Reference” uses `as01hsc410svrb` .

3 Click **Go**.

Visual Studio will connect to the server and search for the WCF service. Once found, the **Services** and **Operations** lists will populate.

4 Provide the namespace. Type `ActivityRef` in the **Namespace** box.

Attention

`ActivityRef` is the recommended namespace, and is used in this documentation. However, any desired namespace can be specified here.

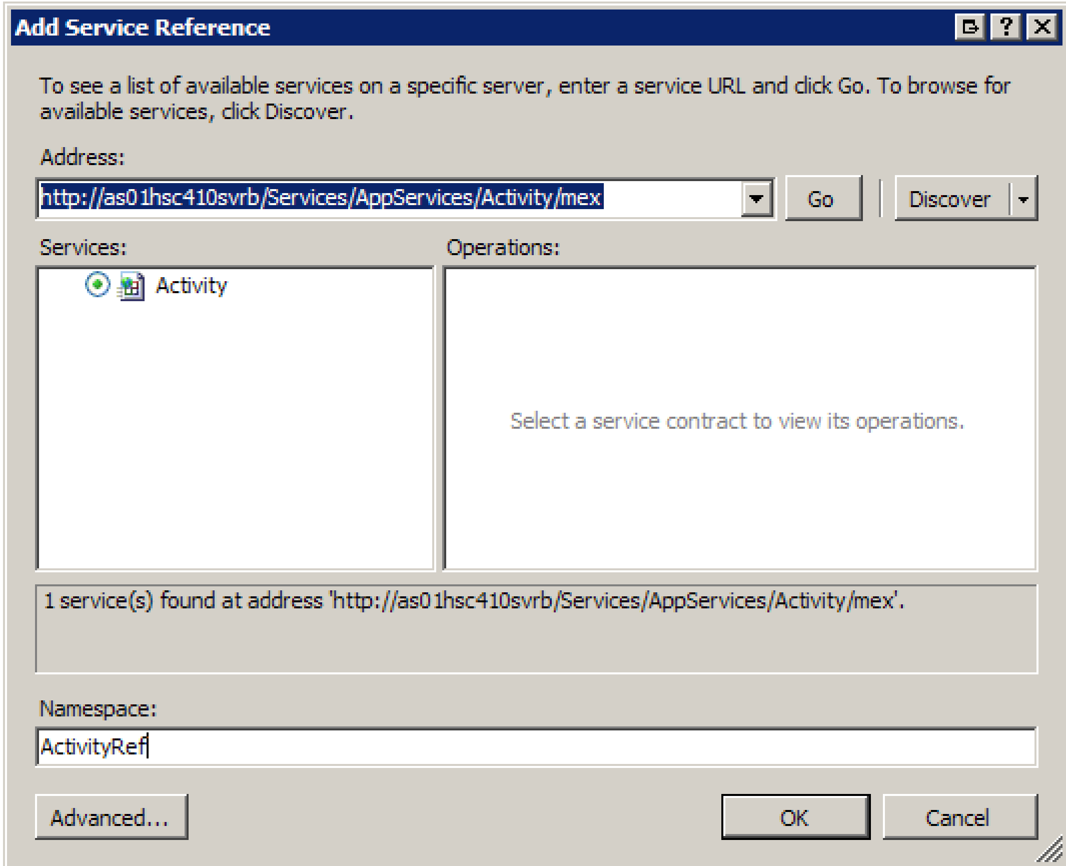


Figure 36: Add Service Reference

5 Click **Advanced**.

The **Service Reference Settings** window appears.

6 Select *System.Collections.Generic.List* from the **Collection type** list.

Selecting this option means that, for the types exposed by the service, any type that is a collection of other types will be contained in a .Net Generic List, as opposed to a .Net Array.

(This tutorial uses a .Net Generic List for collections of objects. However, other collection types can be used as desired.)

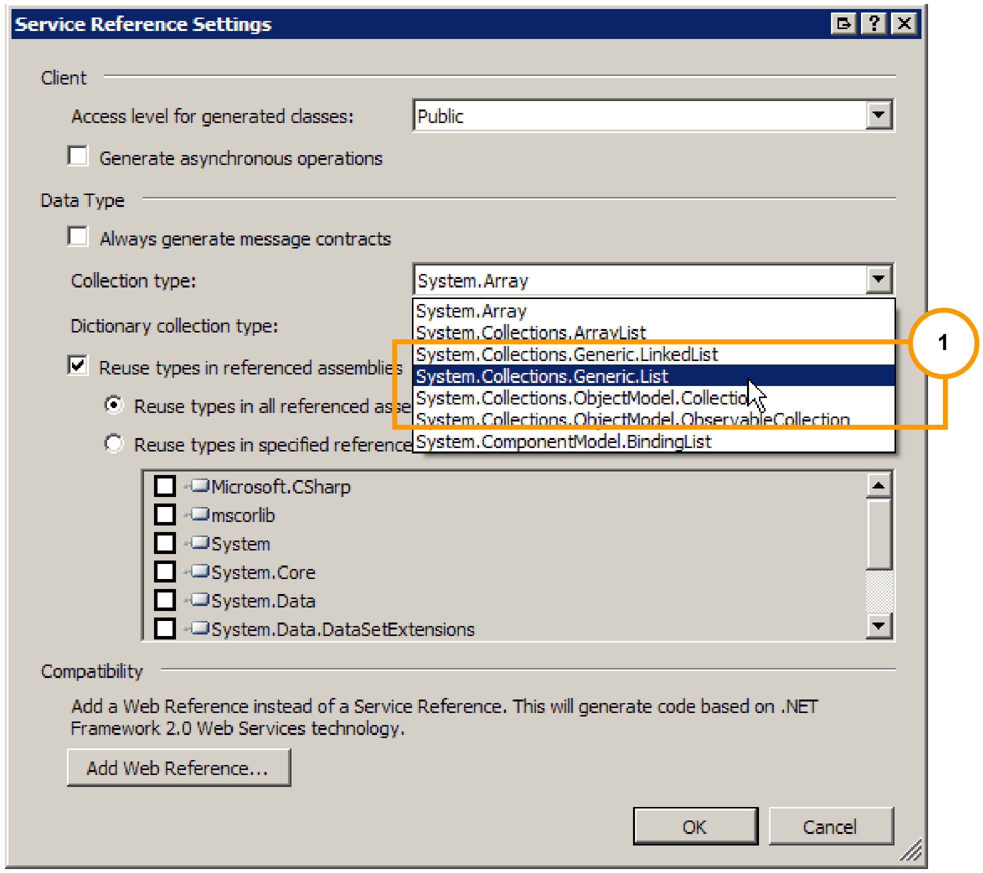


Figure 37: Service Reference Settings

Item	Description
1	System.Collections.Generic.List item.

7 Click **OK**.

The **Service Reference Settings** window closes.

8 Click **OK**.

The **Add Service Reference** window closes and Visual Studio consumes the service into your C# project.

Results

The `ActivityRef` service appears in the **Solution Explorer**.

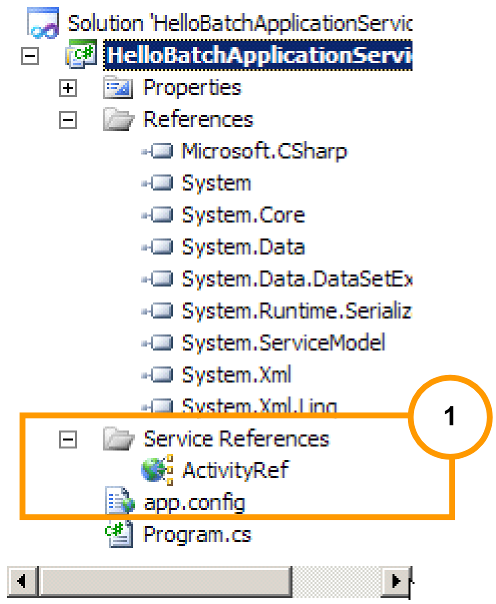


Figure 38: Solution Explorer

Item	Description
1	ActivityRef service in the Solution Explorer.

You have completed the steps to consume the WCF Service.

Next, you will query the IsPrimary operation, located in the Simple tutorial. See "Simple tutorial" on page 394.

Adjusting app.config to support large list operations

After the WCF service has been consumed, the project creates a new configuration file titled *app.config*. This file contains configuration settings for the newly consumed service. You need to adjust some of these settings in order for the project to support operations with large lists.

The following code is an example configuration file that has been modified to support large lists sizes. Compare this with the version that was generated automatically in your tutorial project and spot the differences.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<system.serviceModel>
    <bindings>
        <netTcpBinding>
            <binding
                name="NetTcpEndpoint_IActivity"
                closeTimeout="00:01:00"
                openTimeout="00:01:00"
                receiveTimeout="00:10:00"
                sendTimeout="00:01:00"
                transactionFlow="false"
                transferMode="Buffered"
                transactionProtocol="OleTransactions"
                hostNameComparisonMode="StrongWildcard"
                listenBacklog="10"
                maxBufferPoolSize="20971520"
                maxBufferSize="10485760"
                maxConnections="10"
                maxReceivedMessageSize="10485760">
            <readerQuotas
                maxDepth="32"
                maxStringContentLength="8192"
                maxArrayLength="16384"
                maxBytesPerRead="4096"
                maxNameTableCharCount="16384" />
```

```
            <reliableSession
                ordered="true"
                inactivityTimeout="00:10:00"
                enabled="true" />
            <security
                mode="Transport">
            <transport
                clientCredentialType="Windows"
                protectionLevel="EncryptAndSign" />
            <message clientCredentialType="Windows" />
            </security>
            </binding>
        </netTcpBinding>
    </bindings>
    <behaviors>
        <endpointBehaviors>
            <behavior
                name="Honeywell.Client.Application.Services.ActivityBehavior">
                    <dataContractSerializer maxItemsInObjectGraph="1000000" />
                    <clientCredentials>
                    <windows allowedImpersonationLevel="Impersonation" />
                    </clientCredentials>
            </behavior>
        </endpointBehaviors>
    </behaviors>
    <client>
        <endpoint
            address= "net.tcp://as01hsc410svrb:40200/Services/AppServices/Activity"
            binding="netTcpBinding"
            bindingConfiguration="NetTcpEndpoint_IActivity"
            contract="Ref.AppServices"
            name="NetTcpEndpoint_IActivity"
            behaviorConfiguration= "Honeywell.Client.Application.Services.ActivityBehavior">
        </endpoint>
    </client>
</system.serviceModel>
</configuration>
```

Attention

The address of the server that a Batch Application Services API client will connect to is configured in <client><endpoint>'s address property. In this example, the server name is *as01hsc410svra*. Make sure that you specify the primary server in *app.config*, because calling any Batch Application Services API (with the exception of *IsPrimary*) on a backup server will result in a SOAP Fault.

Prerequisites for a C++ project

You will learn how to create a new C++ project, add directories and dependencies, and copy Batch Application Services resources to your project.

Attention

If you want to create client applications in C++ that use redundancy, you also need to configure the Batch Application Services resources configuration file. For more information, see “Configuring the Batch Application Services Client Component” on page 443.

Disclaimer

Before commencing this tutorial the Batch Application Services Client Component should be installed. (See “Installing the Batch Application Services Client Component” on page 440.) If the Batch Application Services Client Component is not installed, then the directories referred to won't exist and runtime linkage errors may occur.

Related topics

“Creating a new C++ project” on page 383
“Adding directories and dependencies” on page 385
“Copying the Batch Application Services resources” on page 391
“Header file and namespaces” on page 393

Creating a new C++ project

To create a new C++ Console Application project

1. Start Visual Studio.
2. From the File menu, click **New** > **Project**.
   The **New Project** window appears.
3. From the *Installed Templates* list, click **Visual C++** > **Win 32**.
4. From the middle pane, click **Win32 Console Application**.
5. Name your project. Type `HelloBatchApplicationServicesCPP` in the **Name** box.
6. Specify where to save the project. Type `c:\Tutorial` in the **Location** box.
   If you prefer, you can specify an alternate location to save the project.

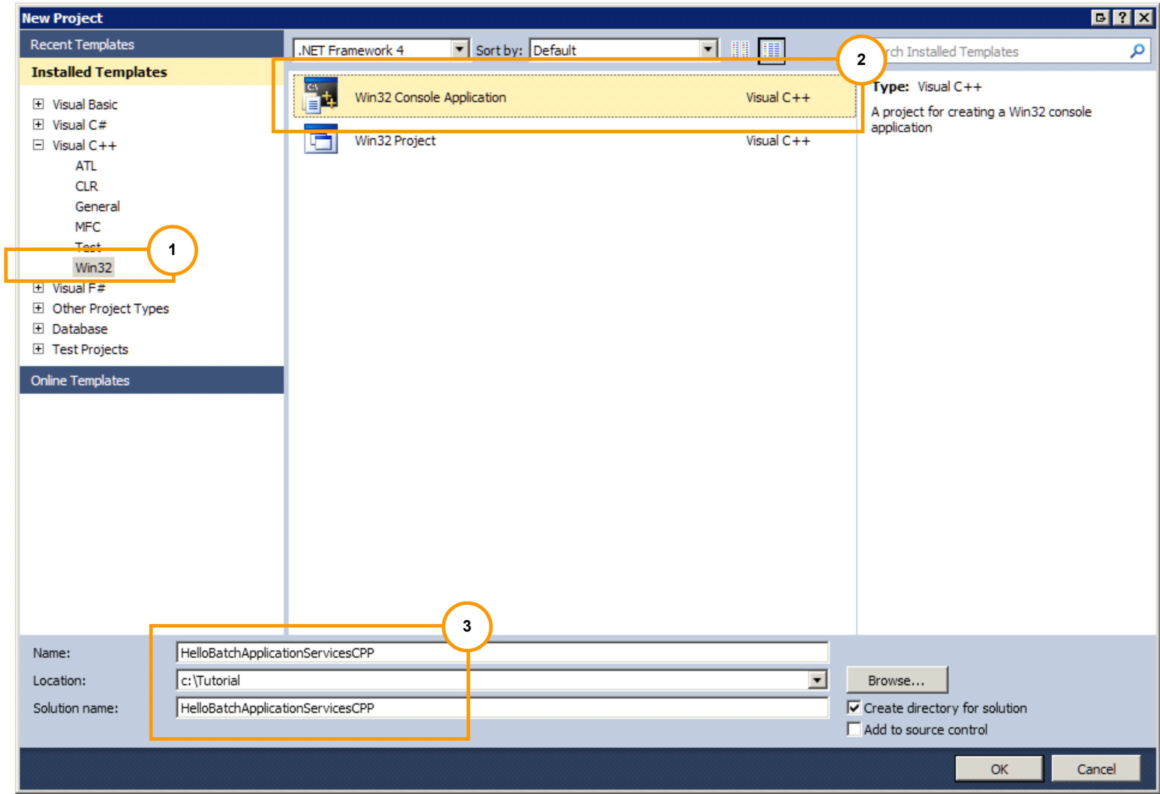


Figure 39: Creating a new C++ project

Item	Description
1	List of installed templates, grouped by programming language.
2	List of available templates for the selected group.
3	File name, location, and Solution name information.

7 Click **OK** to start the **Win32 Application Wizard**.
Visual Studio starts the **Win32 Application Wizard**.

8 On the **Overview** page, click **Next**.

9 On the **Application Settings** page, clear the **Precompiled header** check box.

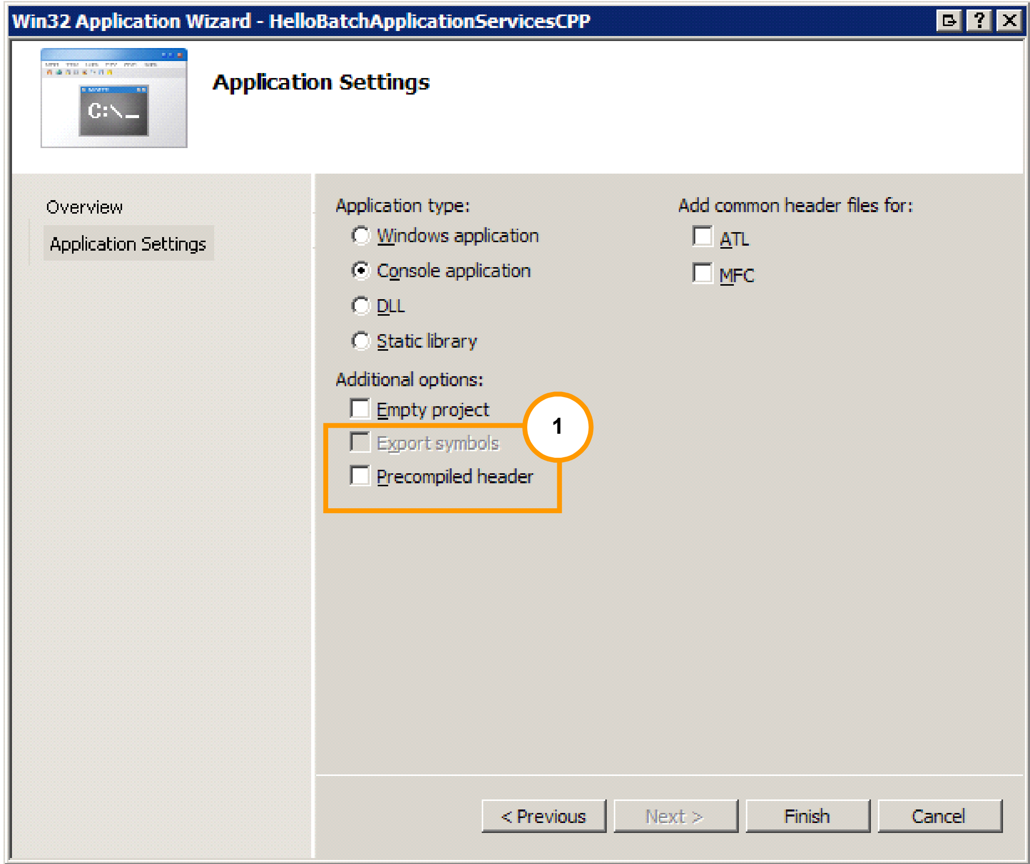


Figure 40: Application Settings

Item	Description
1	Precompiled header check box.

10 Click **Finish**.
Visual Studio creates the project.

Adding directories and dependencies

Now that you have created your project you will configure it so that it complies and links successfully. You will complete the following tasks:

1. First, for the debug builds:
   a. Add an *include directory* (steps 4-6).
   b. Add a *library directory* (steps 7-9).
   c. Add *additional dependencies* (steps 10-12).
2. Then, repeat the steps for the release builds:
   a. Add an *include directory* (steps 4-6).
   b. Add a *library directory* (steps 7-9).
   c. Add *additional dependencies* (steps 10-11, plus step 15).

To add directories and dependencies to your debug and build environments

1 From the *Solution Explorer* list, right-click the **HelloBatchApplicationServicesCPP.cpp** project, and then click **Properties**.

The **Property Pages** window appears.

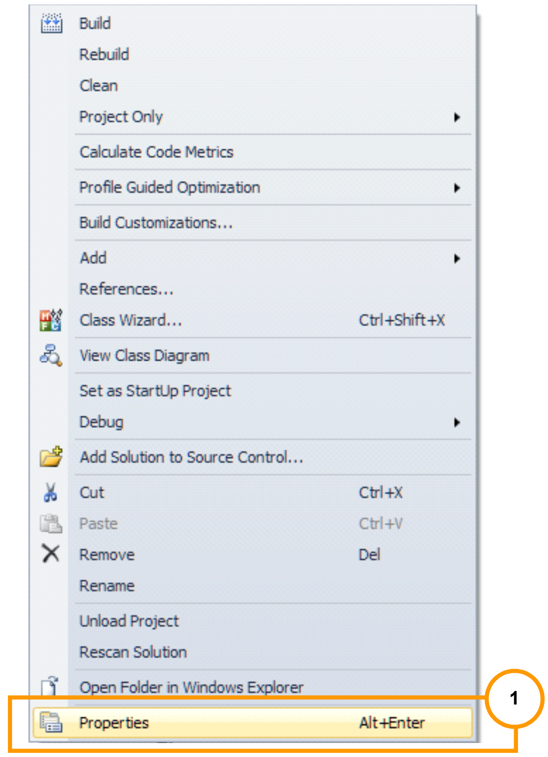


Figure 41: Solution Explorer

Item	Description
1	Properties menu item.

2 In the **Configuration** list, select **Debug**.

3 From the left pane, select **Configuration Properties** > **VC++ Directories**.

4 Next, you will add an *include directory*. From the middle pane, click the drop-down arrow next to **Include Directories**, and then click **Edit**.

The **Include Directories** window appears.

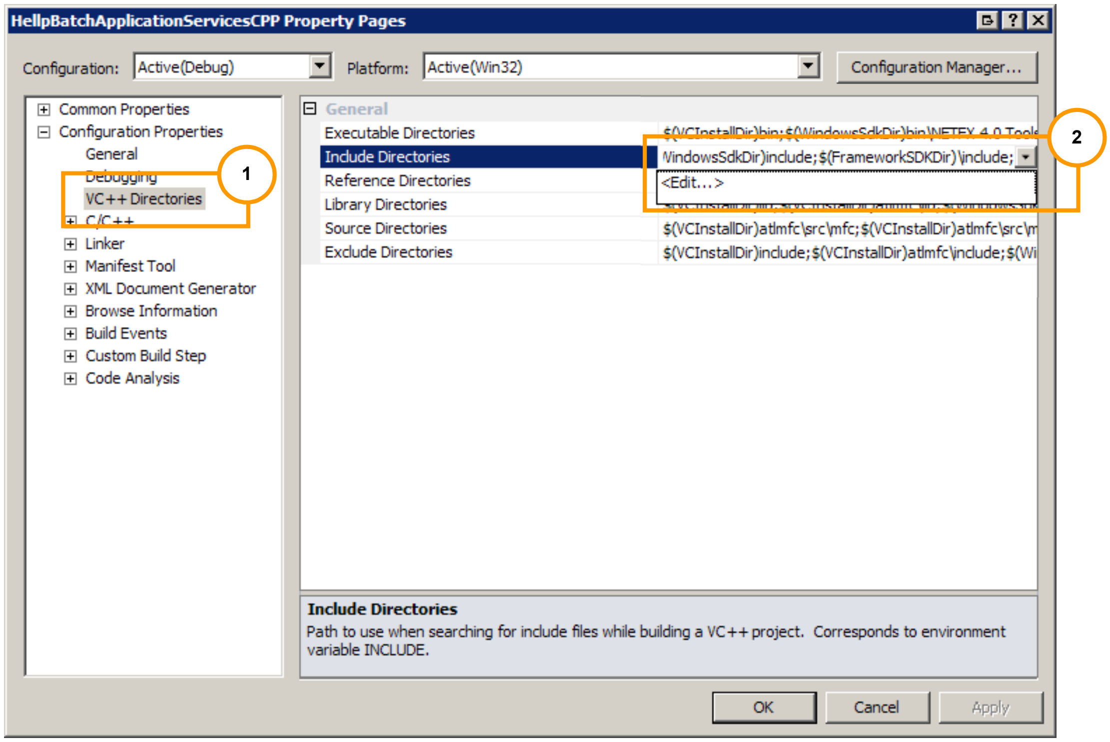


Figure 42: Add Property Pages - Include Directories

Item	Description
1	VC++ Directories in the Configuration Properties folder.
2	Edit item.

5 Click the **New Line** icon to add a new include directory.

6 Type `c:\Program Files (x86)\Honeywell\Client\RedundancyService\include` (assuming the default location was used when installing the Client Side Component) and then click **OK**.

The **Include Directories** window closes.

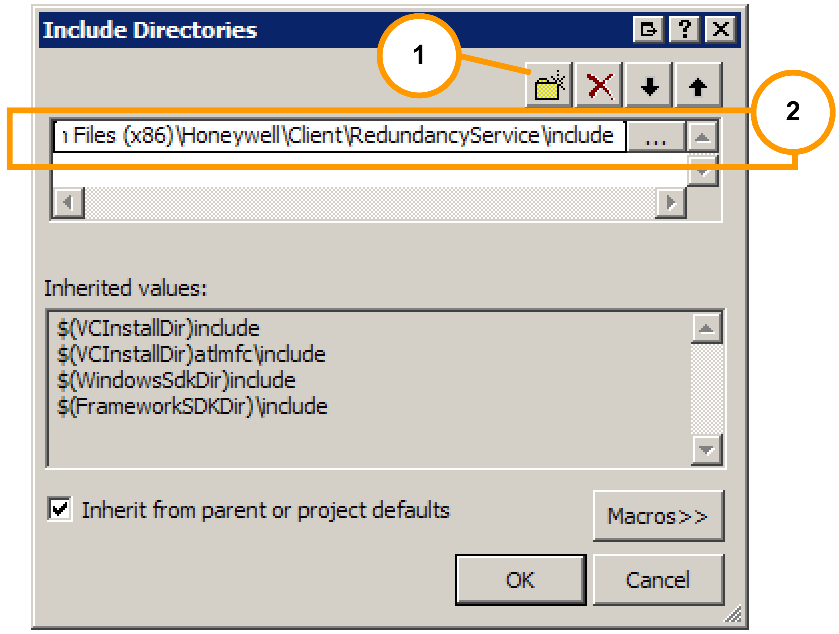


Figure 43: Include Directories

Item	Description
1	New Line icon.
2	File path for the new directory.

7 Next, you will add a *library directory*. From the middle pane of the project property pages dialog, click the drop-down arrow next to **Library Directories**, and then click **Edit**.

The **Library Directories** window appears.

8 Click the **New Line** icon to add a new include directory.

9 Type `c:\Program Files (x86)\Honeywell\Client\RedundancyService\lib` and then click **OK**.

The **Library Directories** window closes.

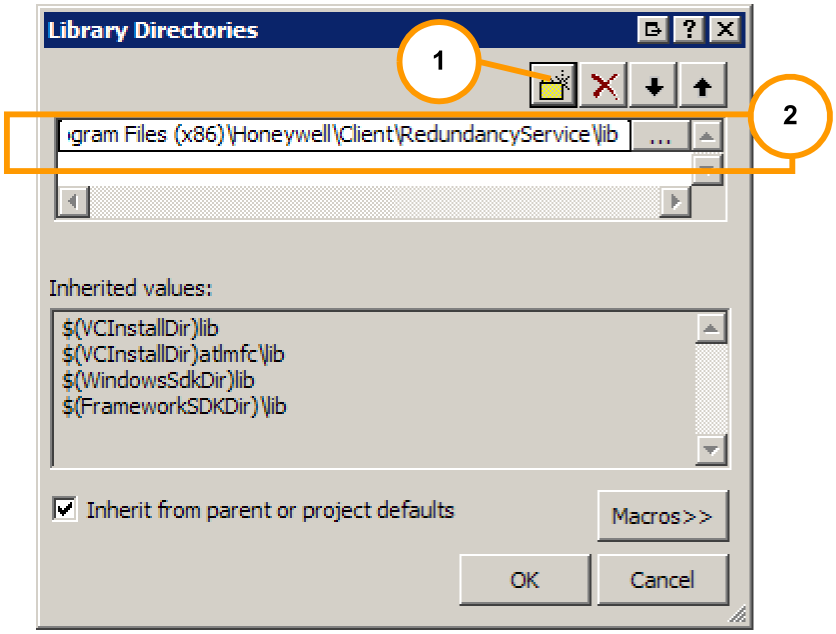


Figure 44: Library Directories

Item	Description
1	New Line icon.
2	File path for the new directory.

10 Next, you will add *additional dependencies*. From the left pane, select **Configuration Properties** > **Linker** > **Input**.

11 From the middle pane, click the drop-down arrow next to **Additional Dependencies**, and then click **Edit**.

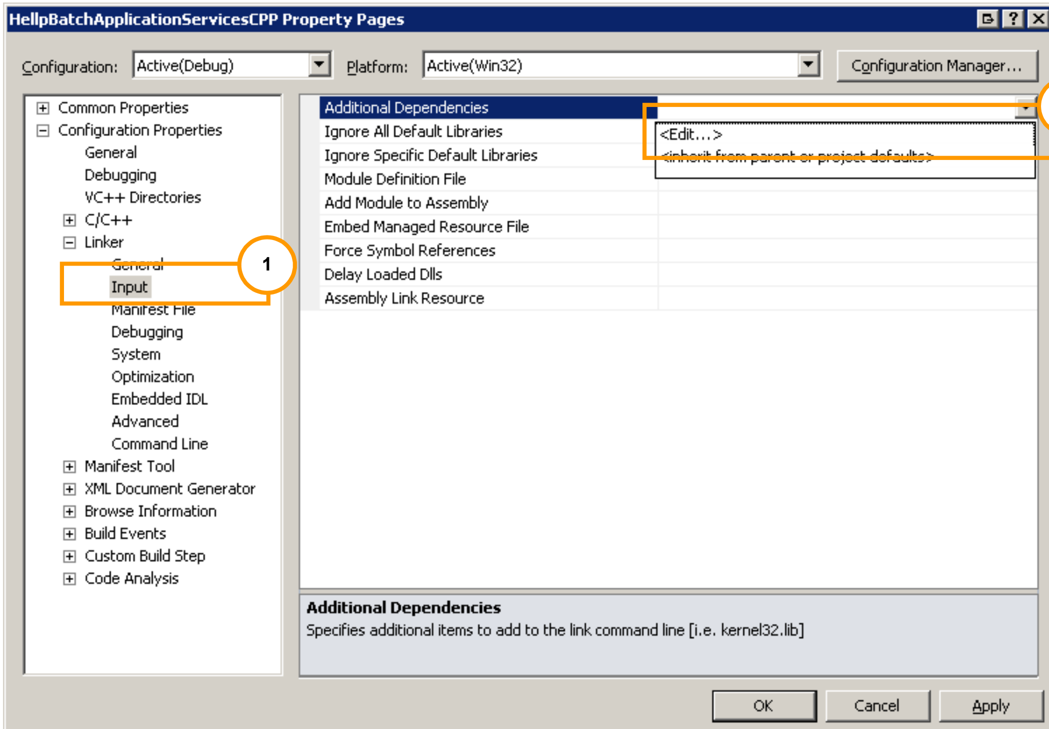


Figure 45: Property Pages

Item	Description
1	Input selection.
2	Edit menu item.

The **Additional Dependencies** dialog appears.

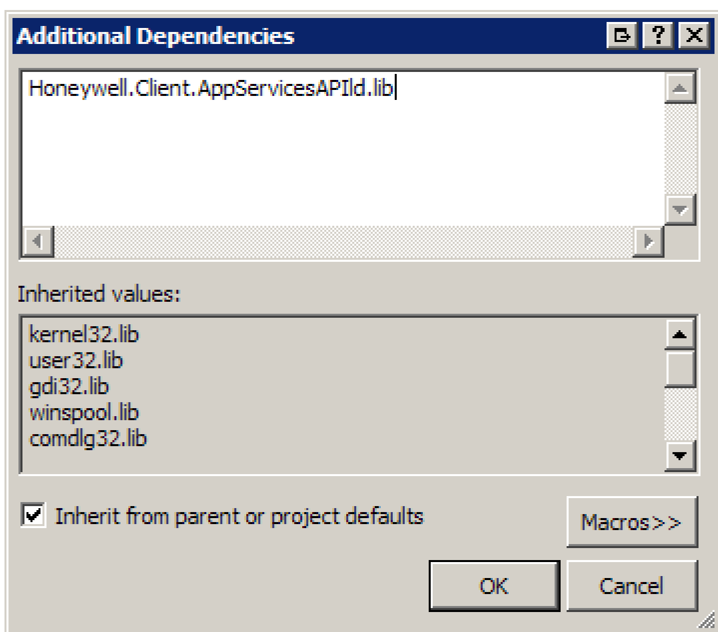


Figure 46: Additional Dependencies

12 Type `Honeywell.Client.AppServicesAPId.lib` , and then click **OK**.

The **Additional Dependencies** dialog closes

13 Now that you have added these three items to the debug environment, you will do the same for the release environment.

In the **Configuration** list, select **Release**.

14 Repeat steps - .

15 Type `Honeywell.Client.AppServicesAPI.lib` , and then click **OK**.

The **Additional Dependencies** window closes

> **Attention**
>
> This file name is very similar to the one you typed in the debug environment. The only difference is the name of the release library does not have the letter `d` (for debug) appended to the end of it. That is, `AppServicesAPI.lib` .

16 Click **OK** to close the project **Property Pages** window.

Results

You have completed the steps to add directories and dependencies to your project.

Next, you will copy the Batch Application Services resources to your project.

Copying the Batch Application Services resources

Now that you have added directories and dependencies to your project you can copy the Batch Application Services resources to the build target directory. You will complete the following tasks:

1. First, for the debug builds, copy the Batch Application Services resources to the build target directory (steps 4-8).
2. Then, repeat the steps for the release builds (steps 9-13).

The Batch Application Services client side DLLs needs to be available to our application when it is executed. A simple way of making these available is to copy them to the directory where our application is executed. Custom logic can be added to the Post-Build Event to copy these DLLs to the build target directory.

To copy the Batch Application Services resources

1 From the *Solution Explorer* tree, right-click the **HelloBatchApplicationServicesCPP** project, and then click **Properties**.

The **Property Pages** window appears.

2 In the **Configuration** list, select **Debug**.

3 From the left pane, select **Configuration Properties** > **Build Events** > **Post-Build Event**.

4 Type `Copy AppServices DLLs` in the **Description** box.

5 From the middle pane, click the drop-down arrow next to **Command Line**, and then click **Edit**.

The **Command Line** window appears.

6 Type the following lines:

```
xcopy /Y "C:\Program Files (x86)\Honeywell\Client\RedundancyService
\Honeywell.Client.AppServicesAPId.dll" "$(TargetDir)"

xcopy /Y "C:\Program Files (x86)\Honeywell\Client\RedundancyService
\Honeywell.Client.AppServicesProxy.dll" "$(TargetDir)

xcopy /Y "C:\Program Files (x86)\Honeywell\Client\RedundancyService
\Honeywell.Client.AppServicesRef.dll" "$(TargetDir)"
```

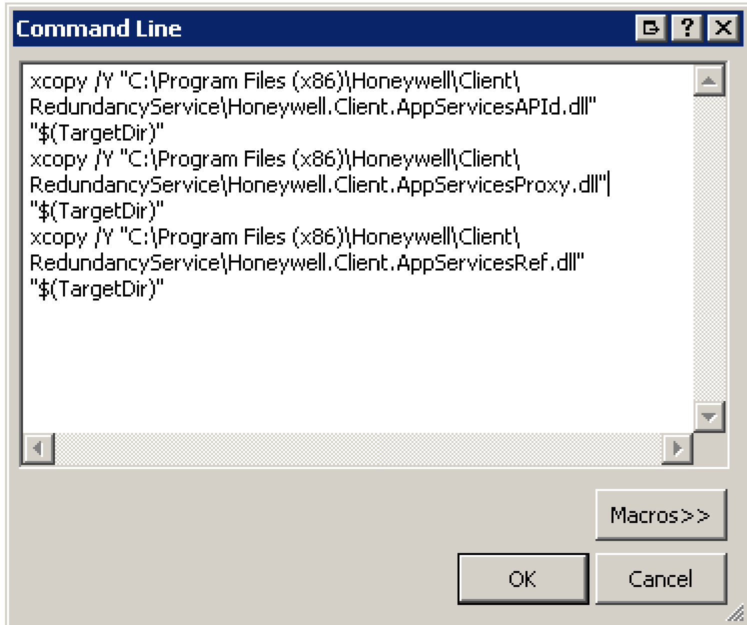


Figure 47: Command Line

7 Click **OK**.
The **Command Line** window closes.

8 On the **Property Pages** window, click **Apply**.

9 Now that you have added the resources to the debug environment, you will do the same for the release environment.
In the **Configuration** list, select **Release**.

10 Repeat steps 3-5.

11 Type the following lines:

```
xcopy /Y "C:\Program Files (x86)\Honeywell\Client\RedundancyService
\Honeywell.Client.AppServicesAPI.dll" "$(TargetDir)"
```

```
xcopy /Y "C:\Program Files (x86)\Honeywell\Client\RedundancyService
\Honeywell.Client.AppServicesProxy.dll" "$(TargetDir)"
```

```
xcopy /Y "C:\Program Files (x86)\Honeywell\Client\ RedundancyService
\Honeywell.Client.AppServicesRef.dll" "$(TargetDir)"
```

Attention

These commands are very similar to the ones you typed in the debug environment. The only difference is the name of the DLL in the first command does not have the letter `d` (for debug) appended to the end of it. That is, `AppServicesAPI.dll`.

12 Click **OK**.
The **Command Line** window closes.

13 On the **Property Pages** window, click **Apply**.

Results

When you build your project, you should see entries similar to the following lines in the build output:

```
1>    PostBuildEvent:
1>    Description: Copy AppServices DLLs
1>    C:\Program Files (x86)\Honeywell\Client\RedundancyService\Honeywell.Client.AppServicesAPI.dll
1>    1 File(s) copied
1>    C:\Program Files (x86)\Honeywell\Client\RedundancyService
\Honeywell.Client.AppServicesProxy.dll
1>    1 File(s) copied1> C:\Program Files (x86)\Honeywell\Client\RedundancyService
```

```
\Honeywell.Client.AppServicesRef.dll
1> 1 File(s) copied
```

You have completed the steps to copy the Batch Application Services resources to your project and have completed the prerequisites.

Header file and namespaces

To access Batch Application Services API declarations and definitions, include *AppServicesAPI.h* in your source code.

To simplify the verbosity of your code you may also choose to include the following namespaces:

```
-Honeywell::Client::Application::Services::API
-Honeywell::Client::Application::Services::XMLTranslate::BatchInformationN
-Honeywell::Client::Application::Services::XMLTranslate::DataContract::FilterN
```

The tutorials samples in the following sections assume the inclusion of the namespaces above.

Next, you can proceed to the simple tutorial.

Simple tutorial

This tutorial shows you the fundamental aspects of the Batch Application Services API. You will complete the following tasks.

Related topics

"Querying the IsPrimary operation" on page 394
"Getting a list of activity entities" on page 396
"Getting more information about an activity entity" on page 397
"Creating an activity" on page 399
"Getting more information about an activity" on page 400
"Configuring an activity before starting it" on page 405
"Running the activity through its life cycle" on page 407
"Monitoring the state of the activity and reacting to state changes" on page 409

Querying the IsPrimary operation

Perform the following steps to determine that communication with Batch Application Services is working correctly.

For C# projects

1 Enter the following code in the *main* method of the console application created in 'Create a New Visual Studio Project.'

```
try
{
    // Create a proxy to communicate with Batch Application Services
    ActivityRef.AppServicesClient proxy =
        new ActivityRef.AppServicesClient();
    // Call IsPrimary as a simple communication test
    bool primary = proxy.IsPrimary();

    if (primary)
        Console.WriteLine("IsPrimary() call succeeded." +
            "The server is running as primary");
    else
        Console.WriteLine("IsPrimary() call succeeded." +
            "The server isn't running as primary");
    Console.WriteLine("Press any key to continue...");
    Console.Read();}catch (Exception ex)
{
    // Something went wrong
    Console.WriteLine("Exception thrown while trying " +
        "to invoke IsPrimary()");
}
```

2 Press the F5 key to compile and run the application.

If the *IsPrimary* operation compiles and runs successfully a message, similar to the following, appears.

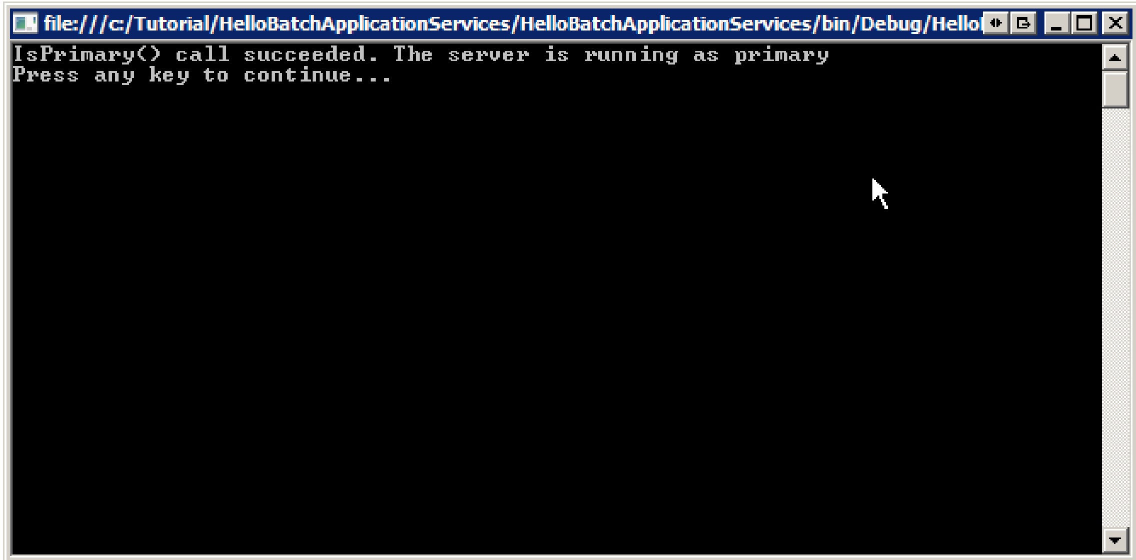


Figure 48: Successful IsPrimary operation

> **Attention**
> The following message also indicates a successful call:
> *IsPrimary() call succeeded. The server is not running as primary*

If the call fails, then the following message appears:

Exception thrown while trying to invoke IsPrimary()

Refer to the “WCF exceptions” on page 373 for guidance on resolving the issue.

You have completed the steps to determine that communication with Batch Application Services is working correctly.

For C++ projects

1 Enter the following code in the Main method of the console application you created in “Creating a new C++ project” on page 383.
Remember to replace the server name in this example (*as01hsc410svrb*) with the host name of the server to which you want to connect.

```
try
{
    // Create a proxy to communicate with Batch Application Services
    AppServicesAPI* proxy = newAppServicesAPI(L"as01hsc410svrb");
    // Call IsPrimary as a simple communication test
    bool primary = proxy->IsPrimary();

    if (primary)
        wcout << "IsPrimary() call succeeded."
            << "The server is running as primary"
            << endl;
    else
        wcout << "IsPrimary() call succeeded."
            << "The server is not running as primary"
            << endl;
}
catch (exception& ex)
{
    // Something went wrong
    wcout << "Exception thrown while trying "
        << "to invoke IsPrimary()"
        << endl;
    wcout << "ex.what() = " << endl << ex.what() << endl;
}

system("pause");
```

```
return
0;
```

2 Press the F5 key to compile and run the application.

If the *IsPrimary* operation compiles and runs successfully a message, similar to the following, appears.

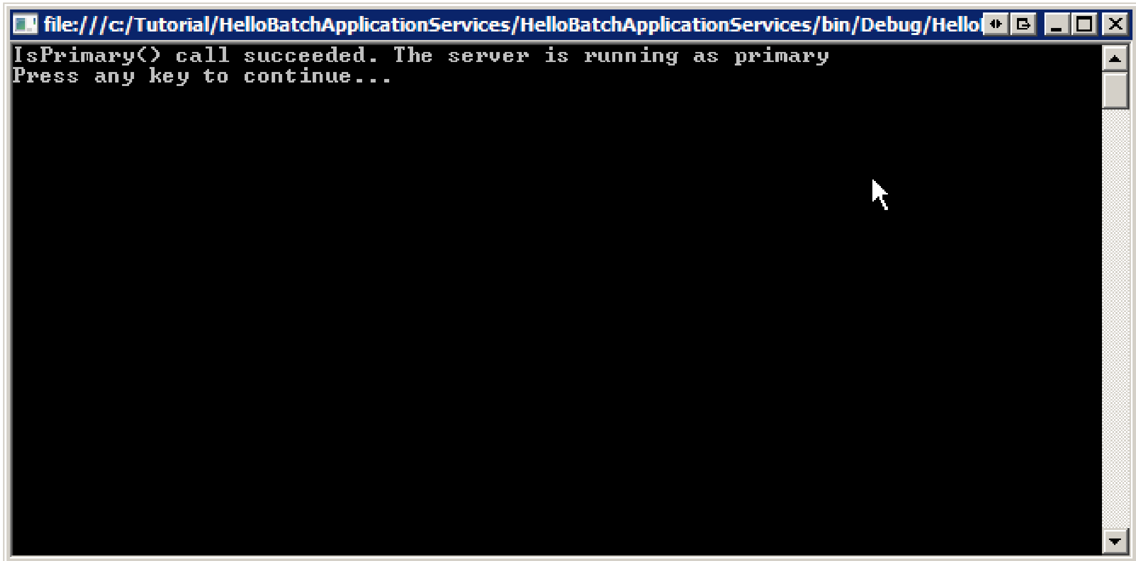


Figure 49: Successful IsPrimary operation

> **Attention**
>
> The following message also indicates a successful call:
>
> *IsPrimary() call succeeded. The server is not running as primary*

If the call fails, then the following message appears:

Exception thrown while trying to invoke IsPrimary()

Refer to “WCF exceptions” on page 373 for guidance on resolving the issue.

You have completed the steps to determine that communication with Batch Application Services is working correctly.

Getting a list of activity entities

Typically, the first operation that needs to be invoked is *GetActivityEntityList*. This operation returns a list of the activity entities configured on a System that are within the calling operators' SOR. Once the list of activity entities is retrieved, you can use other operations, such as *CreateActivity* and *GetActivityEntityMetadata* on one or more individual activity entities.

The following code sample demonstrates retrieving the entire list of activity entities from the server.

Once the *GetActivityEntityList* operation has returned successfully, the *BatchInformationType* will contain a list of *MasterRecipeType* objects. Each *MasterRecipeType* object represents an activity entity. You can use these *MasterRecipeType* objects as arguments to other API operations, as mentioned above.

> **Attention**
>
> This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.
>
> If using redundant servers, the *GetActivityEntityList* and all other Batch Application Services API operations (with the exception of *IsPrimary*) must be invoked against a primary server. If invoked against the backup, an exception will be generated.

Syntax C#

```
// Create a null filter so that all activity entities are returned
ActivityRef.ActivityEntityFilterList actEntFilterList = null;

// Create a Batch Information type to receive the activity entity list
ActivityRef.BatchInformationType batchInfoContainingActivityEntity = null;

// Invoke the GetActivityEntityList operation through the proxy
int opResult = 0;
opResult = proxy.GetActivityEntityList(out batchInfoContainingActivityEntity, actEntFilterList);

// Did the operation complete successfully?
if (opResult == (int)ActivityRef.EReturnCode.SUCCESS)
{
    // Are there any activity entities configured on the system?
    if (batchInfoContainingActivityEntity.MasterRecipe.Count > 0)
    {
        // Extract the details of the first activity entity
        // returned
        ActivityRef.MasterRecipeType firstActEnt =
            batchInfoContainingActivityEntity.MasterRecipe[0];
        string id = firstActEnt.ID.Value;
        string desc = firstActEnt.Description[0].Value;
        string type = firstActEnt.ActivityType.Value;
        DateTimeOffset pointBuildDate =
            firstActEnt.PntbldDate.Value;
        string cluster = firstActEnt.Cluster;
        string asset = firstActEnt.Asset;
        string publicName = firstActEnt.PublicName;
        // ...
    }
}
```

Syntax C++

```
// Create a NULL filter so that all Activity Entities are returned
ActivityEntityFilterList* actEntFilterList = NULL;

// Create a Batch Information type to receive the Activity Entity list
BatchInformationType* batchInfoContainingActivityEntity = NULL;

// Invoke the GetActivityEntityList operation through the proxy
int opResult = proxy->GetActivityEntityList(actEntFilterList, batchInfoContainingActivityEntity);

// Did the operation complete successfully?
if (opResult == AppServicesDefs::SUCCESS)
{
    // Are there any Activity Entities configured on the system?
    if (batchInfoContainingActivityEntity->MasterRecipe.size() > 0)
    {
        // Extract the details of the first Activity Entity returned
        MasterRecipeType* firstActEnt =
            batchInfoContainingActivityEntity->MasterRecipe.front();
        wstring id = firstActEnt->ID->Value;
        wstring desc = firstActEnt->Description[0]->Value;
        wstring type = firstActEnt->ActivityType->Value;
        __int64 pointBuildDate =
            firstActEnt->PntbldDate->Value;
        short    pointBuildDateOffset =
            firstActEnt->PntbldDate->OffsetMinutes;
        wstring cluster = firstActEnt->Cluster;
        wstring asset = firstActEnt->Asset;
        wstring publicName = firstActEnt->PublicName;

        // ...
    }
}
```

Getting more information about an activity entity

The *GetActivityEntityList* operation returns a subset (non-configurable attributes) of the information that can be retrieved for an activity entity. You can use the *GetActivityEntityMetadata* operation to retrieve additional information in the form of metadata for an activity or activity entity. This metadata describes units and unit instances, header, formula, and report parameter names.

The following code sample demonstrates retrieving metadata for an activity entity. It assumes that *batchInfoContainingActivityEntity* is a *BatchInformationType* object returned by *GetActivityEntityList*.

Once the *GetActivityEntityMetadata* operation has returned successfully, the *MasterRecipeType* object (contained within the *BatchInformationType* object) will have had additional metadata added to it.

Attention

This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C#

```
// Retrieve metadata for an activity entity
int opResult = 0;
opResult = proxy.GetActivityEntityMetadata(ref batchInfoContainingActivityEntity);

// Did the operation complete successfully?
if (opResult == (int)ActivityRef.EReturnCode.SUCCESS)
{
    ActivityRef.MasterRecipeType masterRecipeContainingMetadata =
        batchInfoContainingActivityEntity.MasterRecipe[0];

    // Extract the details of the metadata returned
    foreach (ActivityRef.BatchParameterType param in
        masterRecipeContainingMetadata.Formula.Parameter)
    {
        // These values can be displayed, stored, etc.
        string id = param.ID.Value;
        string desc = param.Description.Value;
        string paramType = param.ParameterType.Value;
        string paramSubType = param.ParameterSubType[0].Value;

        // ...

        // Extract the details of each parameter value
        foreach (ActivityRef.BatchValueType value in param.Value)
        {
            // These values can be displayed, stored, etc
                string valString = value.ValueString[0].Value;
                string valDesc = value.Description;
                string valDataIntrp = value.DataInterpretation.Value;
                string valDataType = value.DataType.Value;
                string valUOM = value.UnitOfMeasure.Value;

                // ...
        }
    }
}
```

Syntax C++

```
// Retrieve metadata for an Activity Entity
int opResult = proxy->
    GetActivityEntityMetadata(batchInfoContainingActivityEntity);

// Did the operation complete successfully?
if (opResult == AppServicesDefs::SUCCESS)
{
    MasterRecipeType* masterRecipeContainingMetadata =
        batchInfoContainingActivityEntity->MasterRecipe.front();

    // Extract the details of the metadata returned
    for each (BatchParameterType* param in
        masterRecipeContainingMetadata->Formula->Parameter)
    {
        // These values can be displayed, stored, etc
        wstring id = param->ID->Value;
        wstring desc = param->Description->Value;
        wstring paramType = param->ParameterType->Value;
        wstring paramSubType = param->ParameterSubType.front()->Value;

        // ...

        // Extract the details of each parameter value
        for each (BatchValueType* value in param->Value)
        {
            // These values can be displayed, stored, etc
```

```
            wstring valString = value->ValueString.front()->Value;
            wstring valDesc = value->Description;
            wstring valDataIntrp = value->DataInterpretation->Value;
            wstring valDataType = value->DataType->Value;
            wstring valUOM = value->UnitOfMeasure->Value;

            // ...
        }
    }
}
```

Creating an activity

The *CreateActivity* operation takes a *MasterRecipeType* (representing an activity entity) and returns a *BatchListEntryType* object (representing a newly created activity).

The following code sample demonstrates creating an activity from an activity entity. It assumes that *masterRecipeContainingMetadata* is a *MasterRecipeType* objet within the *BatchInformationType* object returned by *GetActivityEntityList*.

Attention
This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C#

```
// Create a Control Recipe that can be used to create an activity
ActivityRef.ControlRecipeType activityTemplate =
    new ActivityRef.ControlRecipeType();

// Copy activity entity ID from a Master Recipe
activityTemplate.ID = new ActivityRef.IDType();
activityTemplate.ID.Value = masterRecipeContainingMetadata.ID.Value;

// Provide a name for the activity
activityTemplate.BatchID = new ActivityRef.BatchIDType();
activityTemplate.BatchID.Value = "B0001234";

// Wrap the Control Recipe in a Batch Information object
ActivityRef.BatchInformationType batchInfoControlRecipe =
    new ActivityRef.BatchInformationType();
batchInfoControlRecipe.ControlRecipe =
    new List<ActivityRef.ControlRecipeType>(1);
batchInfoControlRecipe.ControlRecipe.Add(activityTemplate);

// Create Batch Information type to receive the newly created activity
ActivityRef.BatchInformationType batchInfoContainingActivity = null;

// Create an activity
int opResult = 0;
opResult = proxy.CreateActivity(out batchInfoContainingActivity,
    batchInfoControlRecipe);

// Did the operation complete successfully?
if (opResult == (int)ActivityRef.EReturnCode.SUCCESS)
{
    ActivityRef.BatchListEntryType activity =
        batchInfoContainingActivity.BatchList[0]
            .BatchListEntry[0];

// Extract the details of the newly created activity
    string id = activity.ID.Value;
    string activityEntityId = activity.RecipeID.Value;
    string activityName = activity.BatchID.Value;
    string owningCluster = activity.Cluster;
    string type = activity.BatchListEntryType1.Value;

    // If this is a Procedure type activity then Value will be 'Other' and
    // OtherValue will be 'Procedure'
    if (type.Equals("Other"))
        type = activity.BatchListEntryType1.OtherValue;

    // ...
}
```

Syntax C++

```
// Create a Control Recipe which can be used to create an Activity
ControlRecipeType* activityTemplate = new ControlRecipeType();

// Copy Activity Entity ID from a Master Recipe
activityTemplate->ID = new IDType();
activityTemplate->ID->Value = masterRecipeContainingMetadata->ID->Value;

// Provide a name for the Activity
activityTemplate->BatchID = new BatchIDType();
activityTemplate->BatchID->Value = L"B0001234";

// Wrap the Control Recipe in a Batch Information object
BatchInformationType* batchInfoControlRecipe =
    new BatchInformationType();
batchInfoControlRecipe->ControlRecipe
    .push_back(activityTemplate);

// Create Batch Information type to receive the newly created Activity
BatchInformationType* batchInfoContainingActivity = NULL;

// Create an Activity
int opResult = 0;
opResult = proxy->CreateActivity(batchInfoControlRecipe,
    batchInfoContainingActivity);

// Did the operation complete successfully?
if (opResult == AppServicesDefs::SUCCESS)
{
    BatchListEntryType* newActivity =
        batchInfoContainingActivity->BatchList
        .front()->BatchListEntry.front();

// Extract the details of the newly created activity
    wstring id = newActivity->ID->Value;
    wstring activityEntityId = newActivity->RecipeID->Value;
    wstring activityName = newActivity->BatchID->Value;
    wstring owningCluster = newActivity->Cluster;
    wstring type = newActivity->BatchListEntryType1->Value;

    // if this is a Procedure type Activity then Value will be 'Other' and
    // OtherValue will be 'Procedure'
    if (type == L"Other")
        type = newActivity->BatchListEntryType1->OtherValue;

    // ...
```

Getting more information about an activity

Returning only one activity

You can use the *GetActivityList* and *GetActivityEntityMetadata* operations to retrieve more information about an activity.

When using the *GetActivityList* operation, you can construct a simple filter so that only the activities of interest are returned.

The following code sample demonstrates using the *GetActivityList* operation to retrieve a single activity with a specific ID (given in the filter). It assumes that activity is a *BatchListEntry* object returned by *CreateActivity*.

Attention

This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C#

```
// Construct a simple filter so that only the activity we are interested in is returned
ActivityRef.ActivityFilterList filterList =
    new ActivityRef.ActivityFilterList();
filterList.TagNameFilter = new ActivityRef.StringFilter();

// Specify the activity ID as the filter criteria
ActivityRef.CriteriaString criStr =
```

```
    new ActivityRef.CriteriaString();
criStr.FilterMode = ActivityRef.EStringFilterModes.Equal;
criStr.CriteriaValue = activity.ID.Value;

// Add criteria to filter
filterList.TagNameFilter.Criteria = criStr;

// Create Batch Information type to receive the list of Activities
ActivityRef.BatchInformationType batchInfoContainingActivities = null;

// Retrieve the list of activities
int opResult = 0;
opResult = proxy.GetActivityList(
    out batchInfoContainingActivities, filterList);

// Did the operation complete successfully?
if (opResult == (int)ActivityRef.EReturnCode.SUCCESS)
{
    // How many activities met our filter criteria? (should be only one)
    int count = batchInfoContainingActivities.BatchList[0]
        .BatchListEntry.Count;
    if (count == 1)
    {
        // Extract the details of this activity
        ActivityRef.BatchListEntryType activityDetails =
            batchInfoContainingActivities.BatchList[0]
            .BatchListEntry[0];

        string id = activityDetails.ID.Value;
        string activityEntityId = activityDetails.RecipeID.Value;
        string activityName = activityDetails.BatchID.Value;
        string owningCluster = activityDetails.Cluster;
        string type = activityDetails.BatchListEntryType1.Value;

        // if this is a Procedure type activity then Value will be 'Other' and OtherValue will be
'Procedure'
        if (type.Equals("Other"))type =
            activityDetails.BatchListEntryType1.OtherValue;

        string desc = activityDetails.Description[0].Value;
        string status = activityDetails.Status.Value;
        string mode = activityDetails.Mode.Value;
        string owningAsset = activityDetails.Asset;
        string publicName = activityDetails.PublicName;
        string modeAttr = activityDetails.ModeAttribute;
        string execStatus = activityDetails.ExecutionStatus;
        string stage = activityDetails.Stage;
        DateTimeOffset createTime =
            activityDetails.CreateTime.Value;

        // If the activity has not started ActualStartTime will be null
        DateTimeOffset startTime =
            (activityDetails.ActualStartTime == null) ?
            DateTimeOffset.MinValue :
                activityDetails.ActualStartTime.Value;

        // If the activity has not ended ActualEndTime will be null
        DateTimeOffset endTime = (activityDetails.ActualEndTime == null) ?
            DateTimeOffset.MinValue :
                activityDetails.ActualEndTime.Value;

        // ...
    }
}
```

Syntax C++

```
// Construct a simple filter so that only the Activity we're
// interested in is returned
ActivityFilterList* filterList = new ActivityFilterList();
filterList->TagNameFilter = new StringFilter();

// Specify the Activity ID as the filter criteria
CriteriaString* criStr = new CriteriaString();
criStr->FilterMode = EStringFilterModes::Equal;
criStr->CriteriaValue = activity->ID->Value;

// Add criteria to filter
filterList->TagNameFilter->Criteria = criStr;

// Create Batch Information type to receive the list of Activities
```

```
BatchInformationType* batchInfoContainingActivities = NULL;

// Retrieve the list of Activities
int opResult = 0;
opResult = proxy->GetActivityList(filterList,
    batchInfoContainingActivities);

// Did the operation complete successfully?
if (opResult == AppServicesDefs)
{
    // How many Activities met our filter criteria? (should only be one)
    int count = batchInfoContainingActivities->BatchList.front()
        ->BatchListEntry.size();

    if (count == 1)
    {
        // Extract the details of this Activity
        BatchListEntryType* activityDetails =
            batchInfoContainingActivities->BatchList.front()
            ->BatchListEntry.front();

        wstring id = activityDetails->ID->Value;
        wstring activityEntityId = activityDetails->RecipeID ->Value;
        wstring activityName = activityDetails->BatchID->Value;
        wstring owningCluster = activityDetails->Cluster;
        wstring type = activityDetails->BatchListEntryType1 ->Value;

        // if this is a Procedure type Activity then Value will
        // be 'Other' and OtherValue will be 'Procedure'
        if (type == L"Other")
            type = activityDetails->BatchListEntryType1
            ->OtherValue;        wstring desc = activityDetails->Description.front() ->Value;
        wstring status = activityDetails->Status->Value;
        wstring mode = activityDetails->Mode->Value;
        wstring owningAsset = activityDetails->Asset;
        wstring publicName = activityDetails->PublicName;
        wstring modeAttr = activityDetails->ModeAttribute;
        wstring execStatus = activityDetails->ExecutionStatus;
        wstring stage = activityDetails->Stage;

        __int64 createTime = activityDetails->CreateTime ->Value;
        short createTimeOffset = activityDetails->CreateTime ->OffsetMinutes;

        // If the Activity hasn't started ActualStartTime will be null
        __int64 startTime = (activityDetails->ActualStartTime == NULL) ?
            0 : activityDetails->ActualStartTime->Value;

        short startTimeOffset = (activityDetails
            ->ActualStartTime == NULL) ?
            0 : activityDetails->ActualStartTime->OffsetMinutes;

        // If the Activity hasn't ended ActualEndTime will be null
        __int64 endTime = (activityDetails->ActualEndTime == NULL) ?
            0 : activityDetails->ActualEndTime->Value;
        short endTimeOffset = (activityDetails
            ->ActualEndTime == NULL) ?
            0 : activityDetails->ActualEndTime->OffsetMinutes;
        // ...
    }
}
```

Retrieving metadata information about an activity

You can also use the *GetActivityEntityMetadata* operation to retrieve information about an activity. The metadata returned by *GetActivityEntityMetadata* describes units and unit instances, header, formula, and report parameter names.

The following code sample demonstrates using the *GetActivityEntityMetadata* operation to retrieve metadata about an activity. It assumes that activity is a *BatchListEntry* object returned by *CreateActivity*.

Attention

This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C#

```
// Create a Master Recipe object to receive the metadata
ActivityRef.MasterRecipeType activityData =
```

```
    new ActivityRef.MasterRecipeType();

// Specify the tag name of the activity
activityData.ID = new ActivityRef.IDType();
activityData.ID.Value = activity.ID.Value;

// Wrap the Master Recipe object in a Batch Information object
ActivityRef.BatchInformationType batchInfoContMetadata =
    new ActivityRef.BatchInformationType();batchInfoContMetadata.MasterRecipe =
    new List<ActivityRef.MasterRecipeType>(1);batchInfoContMetadata.MasterRecipe.Add(activityData);

// Retrieve metadata for this activity
int opResult = 0;opResult = proxy.GetActivityEntityMetadata(
    ref batchInfoContMetadata);

// Did the operation complete successfully?
if (opResult == (int)ActivityRef.EReturnCode.SUCCESS)
{
    activityData = batchInfoContMetadata.MasterRecipe[0];
    // Extract the details of the metadata returned
    foreach (ActivityRef.BatchParameterType param in activityData.Formula.Parameter)
    {
        // These values can be displayed, stored, etc
        string id = param.ID.Value;
        string desc = param.Description.Value;
        string paramType = param.ParameterType.Value;
        string paramSubType = param.ParameterSubType[0].Value;

        // ...

        // Extract the details of each parameter value
        foreach (ActivityRef.BatchValueType value in param.Value)

        {
            // These values can be displayed, stored, etc
            string valString = value.ValueString[0].Value;
            string valDesc = value.Description;
            string valDataIntrp = value.DataInterpretation.Value;
            string valDataType = value.DataType.Value;
            string valUOM = value.UnitOfMeasure.Value;

            // ...
        }
    }
}
```

Syntax C++

```
// Create a Master Recipe object to receive the metadata
MasterRecipeType* activityData = new MasterRecipeType();

// Specify the tag name of the Activity
activityData->ID = new IDType();
activityData->ID->Value = activity->ID->Value;

// Wrap the Master Recipe object in a Batch Information object
BatchInformationType* batchInfoContMetadata = new BatchInformationType();
batchInfoContMetadata->MasterRecipe.push_back(activityData);

// Retrieve metadata for this Activity
int opResult = 0;
opResult = proxy
    ->GetActivityEntityMetadata(batchInfoContMetadata);

// Did the operation complete successfully?
if (opResult == AppServicesDefs::SUCCESS)
{
    activityData = batchInfoContMetadata->MasterRecipe.front();

    // Extract the details of the metadata returned
    for each (BatchParameterType* param in activityData->Formula->Parameter)
    {
        // These values can be displayed, stored, etc
        wstring id = param->ID->Value;
        wstring desc = param->Description->Value;
        wstring paramType = param->ParameterType->Value;
        wstring paramSubType = param->ParameterSubType.front() ->Value;

        // ...

        // Extract the details of each parameter value
```

```
        for each (BatchValueType* value in param->Value)
        {
            // These values can be displayed, stored, etc
            wstring valString = value->ValueString.front()->Value;
            wstring valDesc = value->Description;
            wstring valDataIntrp = value->DataInterpretation ->Value;
            wstring valDataType = value->DataType->Value;
            wstring valUOM = value->UnitOfMeasure->Value;

            // ...
        }
    }
}
```

Retrieving values of point parameters

Once all the parameters for an activity are known, you can use the *GetPointParameterValues* operation to retrieve their values.

The following code sample demonstrates using the *GetPointParameterValues* operation to retrieve the values of pointer parameters. It assumes that *activityData* is a *MasterRecipeType* object populated using *GetActivityEntityMetadata*.

Attention
This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C#

```
// Create a Point Parameter object for requesting the activity's parameter values
ActivityRef.PointParamType pointParam =
    new ActivityRef.PointParamType();
pointParam.PointList = new List<ActivityRef.PointType>(1);
ActivityRef.PointType point = new ActivityRef.PointType();
pointParam.PointList.Add(point);

// Provide the ID of the activity
point.ID = activityData.ID.Value;

// All the parameters belong to one point (the activity)
// Add all the parameters to the Point object
point.Parameter = new List<ActivityRef.BatchParameterType>();
foreach (ActivityRef.BatchParameterType param in
    activityData.Formula.Parameter)
{
    point.Parameter.Add(param);
}

// Retrieve the values of the activity's parameters
int opResult = 0;
opResult = proxy.GetPointParameterValues(ref pointParam,
    (int)ActivityRef.ESubscriptionPeriod.CACHE_READ);

// Did the operation complete successfully?
if (opResult == (int)ActivityRef.EReturnCode.SUCCESS)
{
    // Copy the parameter values back into the activityData object
    List<ActivityRef.BatchParameterType>.Enumerator e =
        activityData.Formula.Parameter.GetEnumerator();
    foreach (ActivityRef.BatchParameterType param in
        pointParam.PointList[0].Parameter)
    {
        e.MoveNext();
        e.Current.Value = param.Value;
    }

// Do something with the parameter values (store, display, etc.)
    foreach (ActivityRef.BatchParameterType param in
        activityData.Formula.Parameter)
    {
        string v = param.Value[0].ValueString[0].Value;

        // ...
    }
}
```

Syntax C++

```
// Create a Point Parameter object for requesting the activity's parameter values
PointParamType* pointParam = new PointParamType();
PointType* point = new PointType();
pointParam->PointList.push_back(point);

// Provide the ID of the activity
point->ID =  activity->ID->Value;
// All the parameters belong to one point (the activity)
// Add all the parameters to the Point object
for each (BatchParameterType* param in activityData->Formula->Parameter)
{
    point->Parameter.push_back(param);
}

// Retrieve the values of the activity's parameters
opResult = 0;
opResult = proxy->GetPointParameterValues(pointParam, AppServicesDefs::SUBSCRIPTION_CACHE_READ);
// Did the operation complete successfully?
if (opResult == AppServicesDefs::SUCCESS)
{

    // Copy the parameter values back into the activityData object
    list<BatchParameterType>::iterator itr = activityData->Formula->Parameter.begin();

    for each (BatchParameterType* param in pointParam->PointList.front()->Parameter)
    {
        (*itr)->Value = param->Value;
        itr++;
    }
    // Do something with the parameter values (store, display, etc->)
    for each (BatchParameterType* param in activityData->Formula->Parameter)
    {
        wstring v = param->Value.front()->ValueString.front()->Value;
        // ...
    }
}
```

Configuring an activity before starting it

Before starting an activity, it would be typical to adjust some of the activity's parameters. These parameters are configured during engineering time and control inputs such as the number of items in a batch and material quantities.

The following code sample demonstrates searching through an activity's parameter list for parameters of interest and adjusting their values. The *SetPointParameterValues* operation is then used to set the new values on the server. It assumes that *activityData* is a *MasterRecipeType* object populated using *GetActivityEntityMetadata*.

Attention

This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C#

```
// Create a Point Parameter object for setting the activity's parameter values
ActivityRef.PointParamType pointParam =
    new ActivityRef.PointParamType();
pointParam.PointList = new List<ActivityRef.PointType>(1);
ActivityRef.PointType point = new ActivityRef.PointType();
point.Parameter = new List<ActivityRef.BatchParameterType>();
pointParam.PointList.Add(point);

// Provide the ID of the activity
point.ID = activityData.ID.Value;

// Adjust the parameters of this activity
foreach (ActivityRef.BatchParameterType param in
    activityData.Formula.Parameter)
{
    // Search for the parameters that need to be adjusted
    // Here we know the activity type is Batch and it has the
    // below parameters configured.
    // However, these are configurable parameters so are not
```

```
    // guaranteed to exist.
    // A more realistic scenario would be these parameters are
    // displayed to a user who adjusts them
    if (param.ID.Value.Equals("BATCHSIZECURR"))
    {
        // This parameter has the description 'Current Batch Size'

        // Provide a new value for this parameter
        param.Value = new List<ActivityRef.BatchValueType>(1);
        param.Value.Add(new ActivityRef.BatchValueType());
        param.Value[0].ValueString =
            new List<ActivityRef.ValueStringType>(1);
        param.Value[0].ValueString.Add( new ActivityRef.ValueStringType()); 20 items in this batch
        param.Value[0].ValueString[0].Value = "20";

        // Add this parameter to the list for writing
        point.Parameter.Add(param);
    }     if (param.ID.Value.Equals("DATA.NUMERIC1_01_VAL.VALUE"))
    {
        // This parameter might have a description like 'Qty water' or 'Qty lime'

        // Provide a new value for this parameter
        param.Value = new List<ActivityRef.BatchValueType>(1);
        param.Value.Add(new ActivityRef.BatchValueType());
        param.Value[0].ValueString =
            new List<ActivityRef.ValueStringType>(1);
        param.Value[0].ValueString.Add(
            new ActivityRef.ValueStringType());
        param.Value[0].ValueString[0].Value = "400"; // 400 mL

        // Add this parameter to the list for writing
        point.Parameter.Add(param);
    }
}

// Set the new values of the activity's parameters
int opResult = 0;
opResult = proxy.SetPointParameterValues(ref pointParam);

// Did the operation complete successfully?
if (opResult == (int)ActivityRef.EReturnCode.SUCCESS)
{
    // Values written successfully
}
```

Syntax C++

```
// Create a Point Parameter object for setting the Activity's parameter values
PointParamType* pointParam = new PointParamType();
PointType* point = new PointType();
pointParam->PointList.push_back(point);

// Provide the ID of the activity
point->ID = activityData->ID->Value;

// Adjust the parameters of this Activity
for each (BatchParameterType* param in
    activityData->Formula->Parameter)
{
    // Search for the parameters that need to be adjusted
    // Here we know the Activity type is Batch and it has the
    // following parameters configured.
    // However, these are configurable parameters so are not
    // guaranteed to exist.
    // A more realistic scenario would be these parameters are
    // displayed to a user who adjusts them
    if (param->ID->Value == L"BATCHSIZECURR")
        {
            // This parameter has the description 'Current Batch Size'

            // Provide a new value for this parameter
            param->Value.push_back(new BatchValueType());
            param->Value.front()->ValueString.push_back( new ValueStringType());
            param->Value.front()->ValueString.front()->Value = L"20"; // 20 items in this batch

            // Add this parameter to the list for writing
            point->Parameter.push_back(param);
        }

        if (param->ID->Value == L"DATA->NUMERIC1_01_VAL->VALUE")
        {
```

```
            // This parameter might have a description like 'Qty water' or 'Qty lime'

            // Provide a new value for this parameter
            param->Value.push_back(new BatchValueType());
            param->Value.front()->ValueString.push_back( new ValueStringType());
            param->Value.front()->ValueString.front()->Value = L"400"; // 400 mL

            // Add this parameter to the list for writing
            point->Parameter.push_back(param);
        }
}

// Set the new values of the Activity's parameters
int opResult = 0;
opResult = proxy->SetPointParameterValues(pointParam);

// Did the operation complete successfully?
if (opResult == AppServicesDefs::SUCCESS)
    // AppServicesDefs::SUCCESS is defined as 0
{
    // Values written successfully
}
```

Running the activity through its life cycle

The *SetPointParameterValues* operation can be used to command an activity through its life cycle. Some of the typical activity commands include:

- *1* (Start)
- *2* (Hold)
- *3* (Stop)
- *4* (Abort)
- *5* (Remove)

The method for sending an activity a command is the same for all commands. The following code demonstrates first checking the state on an activity, then sending it a command to start.

The *BatchListEntryType* object *activity* could be an object returned by a call to *CreateActivity*.

> **Attention**
> This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see "Error scenarios and exception handling" on page 435.

Syntax C#

```
// Create a Point Parameter object for reading the Activity's parameter values
ActivityRef.PointParamType pointParam =
    new ActivityRef.PointParamType();
pointParam.PointList = new List<ActivityRef.PointType>(1);

// Create a Point object
ActivityRef.PointType point = new ActivityRef.PointType();
point.Parameter = new List<ActivityRef.BatchParameterType>();
point.ID = activity.ID.Value;

// Provide the ID of the activity
pointParam.PointList.Add(point);
ActivityRef.BatchParameterType param = null;

param = new ActivityRef.BatchParameterType();
param.ID = new ActivityRef.IDType();
param.ID.Value = "actstate";
point.Parameter.Add(param);

param = new ActivityRef.BatchParameterType();
param.ID = new ActivityRef.IDType();
param.ID.Value = "execsts";
point.Parameter.Add(param);

////////////////////////////////////////////////////////
// Check the activity's current state
int opResult = 0;
```

```
opResult = proxy.GetPointParameterValues(
    ref pointParam,
    (int)ActivityRef.ESubscriptionPeriod.CACHE_READ);

if (opResult != (int)ActivityRef.EReturnCode.SUCCESS)
    return;

string actstate = pointParam.PointList[0].Parameter[0].Value[0].ValueString[0].Value;
string execsts = pointParam.PointList[0].Parameter[1].Value[0].ValueString[0].Value;

// Is the activity in the state we expect
if (!(actstate.Equals("PreExecution") && execsts.Equals("Ok")))
    return;

// Reset point param reference
pointParam = new ActivityRef.PointParamType();
pointParam.PointList = new List<ActivityRef.PointType>(1);

// Create a Point object
point = new ActivityRef.PointType();
point.Parameter = new List<ActivityRef.BatchParameterType>();
point.ID = activity.ID.Value;

// Provide the ID of the activity
pointParam.PointList.Add(point);
param = new ActivityRef.BatchParameterType();
param.ID = new ActivityRef.IDType();
param.ID.Value = "actcommand";
param.Value = new List<ActivityRef.BatchValueType>(1);
param.Value.Add(new ActivityRef.BatchValueType());
param.Value[0].DataType = new ActivityRef.DataTypeType();
param.Value[0].DataType.Value = "short";
param.Value[0].ValueString = new List<ActivityRef.ValueStringType>(1);
param.Value[0].ValueString.Add(new ActivityRef.ValueStringType());
param.Value[0].ValueString[0].Value = "1";

// Start
point.Parameter.Add(param);

/////////////////////////////////////////////////////
// Start the activity
opResult = 0;
opResult = proxy.SetPointParameterValues(ref pointParam);

if (opResult != (int)ActivityRef.EReturnCode.SUCCESS)
    return; // Command failed
```

Syntax C++

```
// Create a Point Parameter object for reading the Activity's parameter
values
PointParamType* pointParam = new PointParamType();

// Create a Point object
PointType* point = new PointType();
point->ID = activity->ID->Value;

// Provide the ID of the activity
pointParam->PointList.push_back(point);
BatchParameterType* param = new BatchParameterType();
param->ID = new IDType();
param->ID->Value = L"actstate";
point->Parameter.push_back(param);
param = new BatchParameterType();
param->ID = new IDType();
param->ID->Value = L"execsts";
point->Parameter.push_back(param);


/////////////////////////////////////////////////////
// Check the activity's current state
opResult = 0;
opResult = proxy->GetPointParameterValues(pointParam, AppServicesDefs::SUBSCRIPTION_CACHE_READ);
if (opResult != AppServicesDefs::SUCCESS)
    return -1;
wstring actstate = pointParam->PointList.front()->Parameter.front()->Value.front()-
>ValueString.front()->Value;
wstring execsts = pointParam->PointList.front()->Parameter.front()->Value.front()-
>ValueString.front()->Value;

// Is the activity in the state we expect
```

```
if (!(actstate == L"PreExecution" && execsts == L"Ok"))
    return -1;

// Reset point param reference
pointParam = new PointParamType();

// Create a Point object
point = new PointType();
point->ID = activity->ID->Value;

// Provide the ID of the activity
pointParam->PointList.push_back(point);
param = new BatchParameterType();
param->ID = new IDType();
param->ID->Value = L"actcommand";
param->Value.push_back(new BatchValueType()); param->Value.front()->DataType = new
    DataTypeType();
param->Value.front()->DataType->Value = L"short";
param->Value.front()->ValueString.push_back(new ValueStringType());
param->Value.front()->ValueString.front()->Value = L"1";

// Start
point->Parameter.push_back(param);

////////////////////////////////////////////////////////
// Start the activity
opResult = 0;
opResult = proxy->SetPointParameterValues(pointParam);
if (opResult != AppServicesDefs::SUCCESS)
    return -1; // Command failed
```

Monitoring the state of the activity and reacting to state changes

The *GetActivityList* and *GetPointParameterValues* operations can be used to monitor the state of an activity.

The following code samples demonstrate two possible techniques for monitoring an activity's *state*. The first uses the *GetActivityList* operation and the second the *GetPointParameterValues* operation.

The *BatchListEntryType* object *activity* could be an object returned by a call to *CreateActivity*.

Attention

This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C# - GetActivityList operation

```
// Construct a filter to isolate the activity we're interested in
ActivityRef.ActivityFilterList filterList =
    new ActivityRef.ActivityFilterList();
filterList.TagNameFilter = new ActivityRef.StringFilter();
ActivityRef.CriteriaString criStr =
    new ActivityRef.CriteriaString();
criStr.FilterMode = ActivityRef.EStringFilterModes.Equal;

// Provide the activity id
criStr.CriteriaValue = activity.ID.Value;
filterList.TagNameFilter.Criteria = criStr;

ActivityRef.BatchInformationType batchInfo = null;
ActivityRef.BatchListEntryType activityData = null;

// Wait for the activity to complete
// Poll the activity once every 2 seconds
while (true)
{
    if (proxy.GetActivityList(out batchInfo, filterList) !=
            (int)ActivityRef.EReturnCode.SUCCESS)
        break;
        // call failed

    // Did any activities match the filter?
    if (batchInfo.BatchList[0].BatchListEntry.Count > 0)
    {
        activityData = batchInfo.BatchList[0].BatchListEntry[0];
        string stage = activityData.Stage;
        string exeStatus = activityData.ExecutionStatus;
```

```
        string status = activityData.Status.Value;

        if (status.Equals("Complete"))
        {
            // React to activity completion

            break;
        }
    }
    else
        break;

    System.Threading.Thread.Sleep(2000);
}
```

Syntax C# - GetPointParameterValues operation

```
// Create a Point Parameter object for setting the Activity's parameter values
ActivityRef.PointParamType pointParam = new ActivityRef.PointParamType();
pointParam.PointList = new List<ActivityRef.PointType>(1);

// Create a Point object
ActivityRef.PointType point = new ActivityRef.PointType();
point.Parameter = new List<ActivityRef.BatchParameterType>();
point.ID = activity.ID.Value;

// Provide the ID of the activity
pointParam.PointList.Add(point);
ActivityRef.BatchParameterType param = null;
param = new ActivityRef.BatchParameterType();
param.ID = new ActivityRef.IDType();
param.ID.Value = "actstate";
point.Parameter.Add(param);

// Wait for the activity to complete
// Poll the activity once every 2 seconds
while (true)
{
    if (proxy.GetPointParameterValues(ref pointParam, 2000) !=
        (int)ActivityRef.EReturnCode.SUCCESS)
            break;

        if (pointParam.PointList[0].Parameter[0].Value[0]
            .ValueString[0].Value.Equals("Complete"))
        {
            // React to activity completion

            break;
        }
        System.Threading.Thread.Sleep(2000);
}
```

Syntax C++ - GetActivityList operation

```
// Construct a filter to isolate the activity we're interested in
ActivityFilterList* filterList = new ActivityFilterList();
filterList->TagNameFilter = new StringFilter();
CriteriaString* criStr = new CriteriaString();
criStr->FilterMode = EStringFilterModes::Equal;

// Provide the activity id
criStr->CriteriaValue = activity->ID->Value;
filterList->TagNameFilter->Criteria = criStr;
BatchInformationType* batchInfo = NULL;
BatchListEntryType* activityData = NULL;

// Wait for the activity to complete
// Poll the activity once every 2 seconds
while (true)
{
    if (proxy->GetActivityList(filterList, batchInfo) != AppServicesDefs::SUCCESS)
        break; // call failed
    // Did any activities match the filter?
    if (batchInfo->BatchList.front()->BatchListEntry.size() > 0)
    {
        activityData = batchInfo->BatchList.front()->BatchListEntry.front();
        wstring stage = activityData->Stage;
        wstring exeStatus = activityData->ExecutionStatus;
        wstring status = activityData->Status->Value;
        if (status == L"Complete")
```

```
        {
            // React to activity completion
        }
    }
}
```

Syntax C++ - GetPointParameterValues operation

```
// Create a Point Parameter object for setting the Activity's parameter values
PointParamType* pointParam = new PointParamType();

// Create a Point object
PointType* point = new PointType();

// Provide the ID of the activity
point->ID = activity->ID->Value;
pointParam->PointList.push_back(point);
BatchParameterType* param = NULL;
param = new BatchParameterType();
param->ID = new IDType();
param->ID->Value = L"actstate";
point->Parameter.push_back(param);

// Wait for the activity to complete
// Poll the activity once every 2 seconds
while (true)
{
    if (proxy->GetPointParameterValues(pointParam, 2000) != AppServicesDefs::SUCCESS)
        break;
    if (pointParam->PointList.front()->Parameter.front()
        ->Value.front()->ValueString.front()->Value == L"Complete")
    {
        // React to activity completion
        break;
    }
    Sleep(2000);
}
```

Intermediate tutorial

This tutorial shows you the more advanced aspects of the Batch Application Services API. You will complete the following tasks:

Related topics

"Filtering activity lists and activity entity lists" on page 412
"Activity filter: Type" on page 413
"Activity filter: Asset" on page 414
"Activity filter: Server Base Name" on page 414
"Activity filter: Batch ID" on page 415
"Activity filter: Public Name" on page 416
"Activity filter: Tag Name" on page 417
"Activity filter: Stage" on page 417
"Activity filter: State" on page 418
"Activity filter: Create Time" on page 419
"Activity filter: Actual Start Time" on page 420
"Activity filter: Actual End Time" on page 421
"Combining multiple criteria in string filters" on page 422
"Nesting criteria for DateTimeFilter" on page 424
"Printing returned activities" on page 426
"Activity entity filter: Type" on page 429
"Activity entity filter: Asset" on page 429
"Activity entity filter: Server Base Name" on page 430
"Activity entity filter: Public Name" on page 431
"Activity entity filter: Point Build Date" on page 431
"Printing returned activity entities" on page 432

Filtering activity lists and activity entity lists

The *GetActivityList* and *GetActivityEntityList* operations support filtering. With filtering active, the server returns only those records that meet the conditions set by the client application. Usually, record sets are smaller, which saves network bandwidth, reduces WCF serialization overhead, and improves performance. This section describes how to construct a filter to use with these operations.

Set up for C++ projects

Before attempting to implement any of these filters, make sure the following using statements are present:

```
using namespace std;using namespace
    Honeywell::Client::Application::Services::XMLTranslate::
    DataContract::FilterN;using namespace
    Honeywell::Client::Application::Services::XMLTranslate::
    BatchInformationN;
```

GetActivityList

You can filter on the following fields when creating a filter to use with the *GetActivityList* operation.

Activity's Field	Description
Type	See "Activity filter: Type" on page 413.

Activity's Field	Description
Asset	Matches activities that belong to this asset of any descendent asset of this asset. See "Activity filter: Asset" on page 414.
Cluster Server Base Name	See "Activity filter: Server Base Name" on page 414.
Batch ID	See "Activity filter: Batch ID" on page 415.
Public Name	See "Activity filter: Public Name" on page 416.
Tag Name	See "Activity filter: Tag Name" on page 417.
Stage	See "Activity filter: Stage" on page 417.
State	See "Activity filter: State" on page 418.
Create Time	See "Activity filter: Create Time" on page 419.
Actual Start Time	See "Activity filter: Actual Start Time" on page 420.
Actual End Time	See "Activity filter: Actual End Time" on page 421.

You can combine and nest filter criteria, and print the returned results.

To...	...see
Combine multiple criteria in a string filter	"Combining multiple criteria in string filters" on page 422.
Nest criteria for a DateTimeFilter	"Nesting criteria for DateTimeFilter" on page 424.
Print the returned activities	"Printing returned activities" on page 426.

GetActivityEntityList

You can filter on the following fields when creating a filter to use with the *GetActivityEntityList* operation.

Activity entity's field	Description
Activity Type	See "Activity entity filter: Type" on page 429.
Asset	See "Activity entity filter: Asset" on page 429.
Cluster Server Base Name	See "Activity entity filter: Server Base Name" on page 430.
Public Name	See "Activity entity filter: Public Name" on page 431.
Point Build Date	See "Activity entity filter: Point Build Date" on page 431.

You can print the returned results.

To...	...see
Print the returned activities	"Printing returned activity entities" on page 432.

Activity filter: Type

The following filter restricts the returned activity list to only those activities that are of type *Batch*.

Syntax C#

```
// Create a simple filter on activity type
static void createActivityFilter_Type(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level Activity Filter List
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a new type filter
```

```
    activityFilterList.TypeFilter = new ActivityRef.ActivityTypeFilter();

    // Specify the filtering mode
    activityFilterList.TypeFilter.FilterMode = ActivityRef.EActTypeFilterModes.Equal;

    // Specify the activity type
    activityFilterList.TypeFilter.ActivityType = ActivityRef.EActivityTypes.Batch;
}
```

Syntax C++

```
// Create a simple filter on activity type
void createActivityFilter_Type( NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level Activity Filter List
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a new Type filter
    pActivityFilterList->TypeFilter = new ActivityTypeFilter();

    // Specify the filtering mode
    pActivityFilterList->TypeFilter->FilterMode = EActTypeFilterModes::Equal;

    // Specify the activity type
    pActivityFilterList->TypeFilter->ActivityType = EActivityTypes::Batch;
}
```

Activity filter: Asset

The following filter restricts the returned activity list to only those activities that have a parent or ancestor asset with a tag name of *Building_XYZ*.

Syntax C#

```
// Create a simple filter on activity asset
static void createActivityFilter_Asset(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a new asset filter
    activityFilterList.AssetFilter = new ActivityRef.ActivityAssetFilter();

    // Specify which asset to filter on
    ActivityRef.CriteriaAsset asset = new ActivityRef.CriteriaAsset();
    asset.Asset = "Building_XYZ";
    activityFilterList.AssetFilter.Criteria = asset;

}
```

Syntax C++

```
// Create a simple filter on activity asset
void createActivityFilter_Asset( NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a new asset filter
    pActivityFilterList->AssetFilter = new ActivityAssetFilter();

    // Specify which asset to filter on
    CriteriaAsset* criAsset = new CriteriaAsset();
    criAsset->Asset = L"Building_XYZ";
    pActivityFilterList->AssetFilter->Criteria = criAsset;
}
```

Activity filter: Server Base Name

The following filter restricts the returned activity list to only those activities that belong to the cluster *ExprR410Srv*. (The actual server names are *ExprR410SrvA* and *ExprR410SrvB*.)

Syntax C#

```
// Create a simple filter on cluster base name
static void createActivityFilter_SrvBaseName(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list activityFilterList = new
ActivityRef.ActivityFilterList();

    // Create a server base name filter
    activityFilterList.ServerBaseNameFilter = new ActivityRef.StringFilter();

    // Create a string criteria to specify the server base name
    ActivityRef.CriteriaString criString = new ActivityRef.CriteriaString();

    // Specify the filter mode
    criString.FilterMode = ActivityRef.EStringFilterModes.Equal;

    // Specify the criteria value
    criString.CriteriaValue = "ExprR410Srv";

    // Assign the string criteria to the string filter
    activityFilterList.ServerBaseNameFilter.Criteria = criString;
}
```

Syntax C++

```
// Create a simple filter on cluster base name
void createActivityFilter_SrvBaseName(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a server base name filter
    pActivityFilterList->ServerBaseNameFilter = new StringFilter();

    // Create a string criteria to specify the server base name
    CriteriaString* pCriStr = new CriteriaString();

    // Specify the filter mode
    pCriStr->FilterMode = EStringFilterModes::Equal;

    // Specify the criteria value
    pCriStr->CriteriaValue = L"ExprR410Srv";

    // Assign the string criteria to the string filter
    pActivityFilterList->ServerBaseNameFilter->Criteria = pCriStr;
}
```

Activity filter: Batch ID

The following filter restricts the returned activity list to only those activities that have a batch ID of *Batch-A0123*.

Syntax C#

```
// Create a simple filter on Batch ID
static void createActivityFilter_BatchID(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a Batch ID filter
    activityFilterList.BatchIDFilter = new ActivityRef.StringFilter();

    // Create a string criteria to specify the Batch ID
    ActivityRef.CriteriaString criString = new ActivityRef.CriteriaString();

    // Specify the filter mode
    criString.FilterMode = ActivityRef.EStringFilterModes.Equal;

    // Specify the criteria value
    criString.CriteriaValue = "Batch-A0123";

    // Assign the string criteria to the string filter
```

```
    activityFilterList.BatchIDFilter.Criteria = criString;
}
```

Syntax C++

```
// Create a simple filter on Batch ID
void createActivityFilter_BatchID(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a batch ID filter
    pActivityFilterList->BatchIDFilter = new StringFilter();

    // Create a string criteria to specify the Server Base Name
    CriteriaString* pCriStr= new CriteriaString();

    // Specify the filter mode
    pCriStr->FilterMode = EStringFilterModes::Equal;

    // Specify the criteria value
    pCriStr->CriteriaValue = L"Batch-A0123";

    // Assign the string criteria to the string filter
    pActivityFilterList->BatchIDFilter->Criteria = pCriStr;
}
```

Activity filter: Public Name

The following filter restricts the returned activity list to only those activities that have a public name of *Resin recipe*.

Syntax C#

```
// Create a simple filter on Public Name
static void createActivityFilter_PublicName(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a public name filter
    activityFilterList.PublicNameFilter = new ActivityRef.StringFilter();

    // Create a string criteria to specify the public name
    ActivityRef. CriteriaString criString = new ActivityRef. CriteriaString();

    // Specify the filter mode
    criString.FilterMode = ActivityRef.EStringFilterModes.Equal;

    // Specify the criteria value
    criString.CriteriaValue = "Resin recipe";

    // Assign the string criteria to the string filter
    activityFilterList.PublicNameFilter.Criteria = criString;
}
```

Syntax C++

```
// Create a simple filter on public name
void createActivityFilter_PublicName(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a public name filter
    pActivityFilterList->PublicNameFilter = new StringFilter();

    // Create a string criteria to specify the public name
    CriteriaString* pCriStr = new CriteriaString();

    // Specify the filter mode
    pCriStr->FilterMode = EStringFilterModes::Equal;

    // Specify the criteria value
    pCriStr->CriteriaValue = L"Resin recipe";
```

```
    // Assign the string criteria to the string filter
    pActivityFilterList->PublicNameFilter->Criteria = pCriStr;
}
```

Activity filter: Tag Name

The following filter restricts the returned activity list to only those activities that have a tag name of *ExprR410Srv:$ACTIVITY_10002F12DA92C*. A server base name must be prepended here; the example here is *ExprR410Srv*.

Syntax C#

```
// Create a simple filter on tag Name
static void createActivityFilter_TagName(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a tag name filter
    activityFilterList.TagNameFilter = new ActivityRef.StringFilter();

    // Create a string criteria to specify the tag name
    ActivityRef.CriteriaString criString = new ActivityRef. CriteriaString();

    // Specify the filter mode
    criString.FilterMode = ActivityRef.EStringFilterModes.Equal;

    // Specify the criteria value
    criString.CriteriaValue = "$ExprR410Srv:$ACTIVITY_10002F12DA92C";

    // Assign the string criteria to the string filter
    activityFilterList.TagNameFilter.Criteria = criString;
}
```

Syntax C++

```
// Create a simple filter on tag name
void createActivityFilter_TagName(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a tag name filter
    pActivityFilterList->TagNameFilter = new StringFilter();

    // Create a string criteria to specify the tag name
    CriteriaString* pCriStr = new CriteriaString();

    // Specify the filter mode
    pCriStr->FilterMode = EStringFilterModes::Equal;

    // Specify the criteria value
    pCriStr->CriteriaValue = L"ExprR410Srv:$ACTIVITY_10002F12DA92C";

    // Assign the string criteria to the string filter
    pActivityFilterList->TagNameFilter->Criteria = pCriStr;
}
```

Activity filter: Stage

The following filter restricts the returned activity list to only those activities that have a stage of *Exec*.

Syntax C#

```
// Create a simple filter on stage
static void createActivityFilter_Stage(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a stage filter
```

```
    activityFilterList.StageFilter = new ActivityRef.StringFilter();

    // Create a string criteria to specify the stage
    ActivityRef.CriteriaString criString = new ActivityRef.CriteriaString();

    // Specify the filter mode
    criString.FilterMode = ActivityRef.EStringFilterModes.Equal;

    // Specify the criteria value
    criString.CriteriaValue = "Executing";

    // Assign the string criteria to the string filter
    activityFilterList.StageFilter.Criteria = criString;
}
```

Syntax C++

```
// Create a simple filter on stage
void createActivityFilter_Stage(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a stage filter
    pActivityFilterList->StageFilter = new StringFilter();

    // Create a string criteria to specify the stage
    CriteriaString* pCriStr = new CriteriaString();

    // Specify the filter mode
    pCriStr->FilterMode = EStringFilterModes::Equal;

    // Specify the criteria value
    pCriStr->CriteriaValue = L"Executing";

    // Assign the string criteria to the string filter
    pActivityFilterList->StageFilter->Criteria = pCriStr;
}
```

Activity filter: State

The following filter restricts the returned activity list to only those activities that have a state of *Running*.

Syntax C#

```
// Create a simple filter on state
static void createActivityFilter_State(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a state filter
    activityFilterList.StateFilter = new ActivityRef.StringFilter();

    // Create a string criteria to specify the state
    ActivityRef.CriteriaString criString = new ActivityRef.CriteriaString();

    // Specify the filter mode
    criString.FilterMode = ActivityRef.EStringFilterModes.Equal;

    // Specify the criteria value
    criString.CriteriaValue = "Running";

    // Assign the string criteria to the string filter
    activityFilterList.StageFilter.Criteria = criString;
}
```

Syntax C++

```
// Create a simple filter on state
void createActivityFilter_State(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();
```

```
    // Create a state filter
    pActivityFilterList->StateFilter = new StringFilter();

    // Create a string criteria to specify the state
    CriteriaString* pCriStr = new CriteriaString();

    // Specify the filter mode
    pCriStr->FilterMode = EStringFilterModes::Equal;

    // Specify the criteria value
    pCriStr->CriteriaValue = L"Running";

    // Assign the string criteria to the string filter
    pActivityFilterList->StageFilter->Criteria = pCriStr;
}
```

Activity filter: Create Time

Attention

For information about specifying the UTC offset for a Create Time filter, see “Nesting criteria for DateTimeFilter” on page 424.

The following filter restricts the returned activity list to only those activities that were created in the past hour.

The example below assumes the client and server are in the same time zone. As such, the call to `TimeZone.CurrentTimeZone` returns the time zone of the server. This method will protect against errors around DST, since the actual `DateTime` value is taken into consideration when calculating the UTC offset.

Syntax C#

```
// Create a simple filter on create time
static void createActivityFilter_CreateTime(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a CreateTime filter
    activityFilterList.CreateTimeFilter = new ActivityRef.DateTimeFilter();

    // Create a datetime criteria to specify the Create Time
    ActivityRef.CriteriaDateTime dateTimeCri = new ActivityRef.CriteriaDateTime();

    // Specify the filter mode
    dateTimeCri.FilterMode = ActivityRef.EDateTimeFilterModes.GreaterThan;

    // Assume we are in the same time zone as the server
    TimeZone localTimeZone = TimeZone.CurrentTimeZone;

    // Specify the criteria value (Activities created in the last hour)
    dateTimeCri.CriteriaValue = DateTime.Now.AddHours(-1.0);

    // Get UTC offset in minutes to disambiguate the Date Time value
    dateTimeCri.OffsetMinutes = (short)localTimeZone.GetUtcOffset
        (dateTimeCri.CriteriaValue).TotalMinutes;

    // Assign the datetime criteria to the datetime filter
    activityFilterList.CreateTimeFilter.Criteria = dateTimeCri;
}
```

Syntax C++

```
// Create a simple filter on create time
void createActivityFilter_CreateTime(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a create time filter
    pActivityFilterList->CreateTimeFilter = new DateTimeFilter();

    // Create a DateTime criteria to specify the create time
    CriteriaDateTime* pDtCriActual = new CriteriaDateTime();

    // Specify the filter mode
```

```
    pDtCriActual->FilterMode = EDateTimeFilterModes::GreaterThan;

    // Create new SYSTEMTIME and FILETIME objects and set all members to zero
    SYSTEMTIME stNow, stOneHourAgo;
    FILETIME ftNow;
    ZeroMemory(&stNow, sizeof(SYSTEMTIME));
    ZeroMemory(&stOneHourAgo, sizeof(SYSTEMTIME));
    ZeroMemory(&ftNow, sizeof(FILETIME));

    // Get the current time from the system in UTC
    GetSystemTime(&stNow);

    // Convert to local time
    SystemTimeToTzSpecificLocalTime(NULL, &stNow, &stNow);

    // Convert to FILETIME
    SystemTimeToFileTime(&stNow, &ftNow);

    // Convert to a large integer to perform arithmetic
    LARGE_INTEGER liDateTime;
    liDateTime.HighPart = ftNow.dwHighDateTime;
    liDateTime.LowPart = ftNow.dwLowDateTime;

    // Roll back an hour
    // 100 nanosecond blocks * milliseconds *
    // seconds * minutes * hours
    // 10,000 * 1,000 * 60 * 60 * 1 = 36,000,000,000
    liDateTime.QuadPart -= 36000000000;

    // Convert back to SYSTEMTIME
    FileTimeToSystemTime((FILETIME*)&liDateTime, &stOneHourAgo);

    // Specify the criteria value
    setDateTimeCriteriaValue(&stOneHourAgo, pDtCriActual);

    // Assign the DateTime criteria to the DateTime filter
    pActivityFilterList->CreateTimeFilter->Criteria = pDtCriActual;
}
```

Activity filter: Actual Start Time

The following filter restricts the returned activity list to only those activities that were started on or after 9:00 AM today.

In this example the client and server are assumed to be in the same time zone.

Syntax C#

```
// Create a simple filter on start time
static void createActivityFilter_StartTime(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a StartTime filter
    activityFilterList.StartTimeFilter = new ActivityRef.DateTimeFilter();

    // Create a datetime criteria to specify the Start Time
    ActivityRef.CriteriaDateTime dateTimeCri = new ActivityRef.CriteriaDateTime();

    // Specify the filter mode
    dateTimeCri.FilterMode = ActivityRef.EDateTimeFilterModes.GreaterThanOrEqual;

    // Assume we are in the same time zone as the server
    TimeZone localTimeZone = TimeZone.CurrentTimeZone;

    // Specify the criteria value
    // (Activities started on or after 9:00 AM today)
    DateTime n = DateTime.Now;
    DateTime thisMorning = new DateTime(n.Year, n.Month, n.Day, hour:9, minute:0, second:0,
        kind:n.Kind);
    dateTimeCri.CriteriaValue = thisMorning;

    // Get UTC offset in minutes to disambiguate the Date Time value
    dateTimeCri.OffsetMinutes = (short)localTimeZone.GetUtcOffset
        (dateTimeCri.CriteriaValue).TotalMinutes;

    // Assign the datetime criteria to the datetime filter
```

```
    activityFilterList.StartTimeFilter.Criteria = dateTimeCri;
}
```

Syntax C++

```
// Create a simple filter on start time
void createActivityFilter_StartTime(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a create time filter
    pActivityFilterList->StartTimeFilter = new DateTimeFilter();

    // Create a DateTime criteria to specify the create time
    CriteriaDateTime* pCriDT = new CriteriaDateTime();

    // Specify the filter mode
    pCriDT->FilterMode = EDateTimeFilterModes::GreaterThanOrEqual;

    // Get the local system time
    SYSTEMTIME st;
    GetLocalTime(&st);

    // Adjust to 9:00 AM
    st.wHour = 9;
    st.wMinute = 0;
    st.wSecond = 0;
    st.wMilliseconds = 0;

    // Specify the criteria value
    setDateTimeCriteriaValue(&st, pCriDT);

    // Assign the DateTime criteria to the DateTime filter
    pActivityFilterList->CreateTimeFilter->Criteria = pCriDT;
}
```

Activity filter: Actual End Time

The following filter restricts the returned activity list to only those activities that ended before 5:00 PM today.

In this example the client and server are assumed to be in the same time zone.

Syntax C#

```
// Create a simple filter on end time
static void createActivityFilter_EndTime(ref ActivityRef.ActivityFilterList activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create an EndTime filter
    activityFilterList.EndTimeFilter = new ActivityRef.DateTimeFilter();

    // Create a datetime criteria to specify the end time
    ActivityRef.CriteriaDateTime criDateTime= new ActivityRef.CriteriaDateTime();

    // Specify the filter mode
    criDateTime.FilterMode = ActivityRef.EDateTimeFilterModes.LessThan;

    // Assume we are in the same time zone as the server
    TimeZone localTimeZone = TimeZone.CurrentTimeZone;

    // Specify the criteria value
    // (Activities ended before 5:00 PM today)
    DateTime n = DateTime.Now;
    DateTime thisAfternoon =
        new DateTime(n.Year, n.Month, n.Day,
        hour: 17, minute: 0, second: 0, kind: n.Kind);
        dateTimeCri.CriteriaValue = thisAfternoon;

    // Get UTC offset in minutes to disambiguate the Date Time
    // value    dateTimeCri.OffsetMinutes =
        (short)localTimeZone.GetUtcOffset(
            criDateTime.CriteriaValue).TotalMinutes;

    // Assign the datetime criteria to the datetime filter
```

```
    activityFilterList.EndTimeFilter.Criteria = dateTimeCri;
}
```

Syntax C++

```
// Create a simple filter on End Time
void createActivityFilter_EndTime(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level Activity Filter List
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a Create Time filter
    pActivityFilterList->EndTimeFilter = new DateTimeFilter();

    // Create a DateTime criteria to specify the end time
    CriteriaDateTime* pCriDT = new CriteriaDateTime();

    // Specify the filter mode
    pCriDT->FilterMode = EDateTimeFilterModes::LessThan;

    // Get the local system time
    SYSTEMTIME st;
    GetLocalTime(&st);

    // Adjust to 5:00 PM
    st.wHour = 17;
    st.wMinute = 0;
    st.wSecond = 0;
    st.wMilliseconds = 0;

    // Specify the criteria value
    setDateTimeCriteriaValue(&st, pCriDT);

    // Assign the DateTime criteria to the DateTime filter
    pActivityFilterList->CreateTimeFilter->Criteria = pCriDT;
}
```

Combining multiple criteria in string filters

There may be times when you need to combine multiple criteria in a string filter. You can specify a recursive criteria structure, built with *CriteriaAnd*, *CriteriaOr*, and *CriteriaString* elements. The recursive structure is rooted at a *CriteriaOr* element (*rootStrCriOr*), which contains four *CriteriaString* leaves (*strCriAct1* through *strCriAct4*). “Figure 50: Recursive criteria structure StringFilter” shows the recursive structure for *StringFilter*.

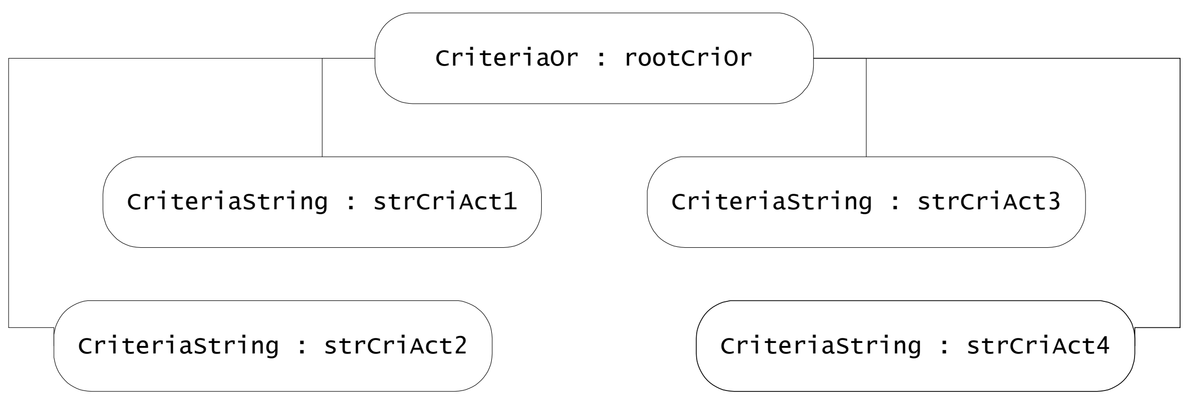


Figure 50: Recursive criteria structure StringFilter

The following example restricts the returned activity list to only those activities that have a State of either *Running*, *Pausing*, *Holding*, or *Stopping*.

The Boolean logic equivalent is: *(State == "Running") OR (State == "Pausing") OR (State == "Holding") OR (State == "Stopping')*.

Syntax C#

```
// Create a filter on State using multiple criteria
static void createActivityFilter_StateMultiple(ref ActivityRef.ActivityFilterList
activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a state filter
    activityFilterList.StateFilter = new ActivityRef.StringFilter();

    // Create the string criteria for this filter
    ActivityRef.CriteriaOr rootStrCriOr = new ActivityRef.CriteriaOr();
    ActivityRef.CriteriaString strCriAct1 = new ActivityRef.CriteriaString();
    ActivityRef.CriteriaString strCriAct2 = new ActivityRef.CriteriaString();
    ActivityRef.CriteriaString strCriAct3 = new ActivityRef.CriteriaString();
    ActivityRef.CriteriaString strCriAct4 = new ActivityRef.CriteriaString();

    // Specify criteria 1 details
    strCriAct1.FilterMode = ActivityRef.EStringFilterModes.Equal; strCriAct1.CriteriaValue =
"Running";

    // Specify criteria 2 details
    strCriAct2.FilterMode = ActivityRef.EStringFilterModes.Equal; strCriAct2.CriteriaValue =
"Pausing";

    // Specify criteria 3 details
    strCriAct3.FilterMode = ActivityRef.EStringFilterModes.Equal; strCriAct3.CriteriaValue =
"Holding";

    // Specify criteria 4 details
    strCriAct4.FilterMode = ActivityRef.EStringFilterModes.Equal;     strCriAct4.CriteriaValue =
"Stopping";

    // Add the CriteriaString to the criteria or
    rootStrCriOr.CriteriaList = new List<ActivityRef.Criteria>();
    rootStrCriOr.CriteriaList.Add(strCriAct1);
    rootStrCriOr.CriteriaList.Add(strCriAct2);
    rootStrCriOr.CriteriaList.Add(strCriAct3);
    rootStrCriOr.CriteriaList.Add(strCriAct4);

    // Assign the root string criteria to the string filter
    activityFilterList.StageFilter.Criteria = rootStrCriOr;
}
```

Syntax C++

```
// Create a filter on State using multiple criteria
void createActivityFilter_StateMultiple(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a state filter
    pActivityFilterList->StateFilter = new StringFilter();

    // Create the string criteria for this filter
    CriteriaOr* pRootCriOr = new CriteriaOr();
    CriteriaString* pCriStr1 = new CriteriaString();
    CriteriaString* pCriStr2 = new CriteriaString();
    CriteriaString* pCriStr3 = new CriteriaString();
    CriteriaString* pCriStr4 = new CriteriaString();

    // Specify criteria 1 details
    pCriStr1->FilterMode = EStringFilterModes::Equal;
    pCriStr1->CriteriaValue = L"Running";

    // Specify criteria 2 details
    pCriStr2->FilterMode = EStringFilterModes::Equal;
    pCriStr2->CriteriaValue = L"Pausing";

    // Specify criteria 3 details
    pCriStr3->FilterMode = EStringFilterModes::Equal;
    pCriStr3->CriteriaValue = L"Holding";

    // Specify criteria 4 details
    pCriStr4->FilterMode = EStringFilterModes::Equal;
    pCriStr4->CriteriaValue = L"Stopping";
```

```
    // Add the criteria actuals to the criteria or
    pRootCriOr->CriteriaList.push_back(pCriStr1);
    pRootCriOr->CriteriaList.push_back(pCriStr2);
    pRootCriOr->CriteriaList.push_back(pCriStr3);
    pRootCriOr->CriteriaList.push_back(pCriStr4);

    // Assign the root string criteria to the string filter
    pActivityFilterList->StageFilter->Criteria = pRootCriOr;
}
```

Nesting criteria for DateTimeFilter

There may be times when you need to nest *DateTimeFilter* objects. You can specify a recursive criteria structure, built with *CriteriaAnd*, *CriteriaOr*, and *CriteriaDateTime* elements. The recursive structure is rooted at a *CriteriaOr* element (*rootStrCriOr*), which contains two *CriteriaAnd* elements (*dtCriAnd1* and *dtCriAct2*). The *CriteriaAnd* elements each contain two *CriteriaDateTime* elements (*dtCriAct1* through *dtCriAct4*). “Figure 51: Nested DateTimeFilter” shows the recursive structure for *DateTimeFilter*.

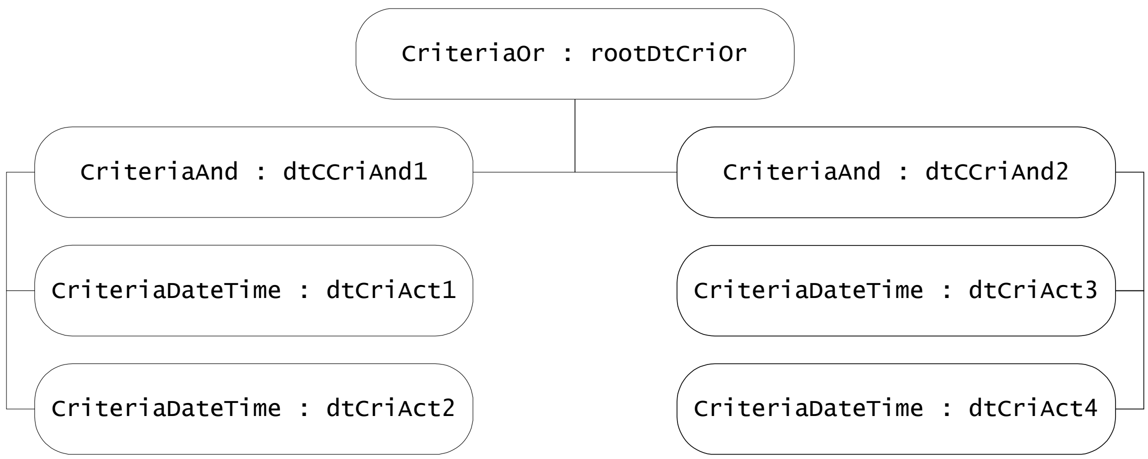


Figure 51: Nested DateTimeFilter

The following example restricts the returned activity list to only those activities that have a Create Time of either between 8:00 AM and 9:00 AM or between 4:00 PM and 5:00 PM; all on June 30, 2012.

The following time line helps to demonstrate the example.

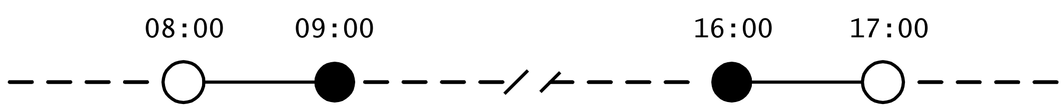


Figure 52: Time line

The Boolean logic equivalent is: *(((CreateTime > 8:00 AM 2011-6-30) AND (CreateTime <= 9:00 AM 2011-6-30)) OR ((CreateTime >= 4:00 PM 2011-6-30) AND (CreateTime < 5:00 PM 2011-6-30)))*.

This example assumes that the client and server are in the same time zone.

Syntax C#

```
// Create a filter on Create Time using nested criteria
static void createActivityFilter_CreateTimeNested(ref ActivityRef.ActivityFilterList
activityFilterList)
{
    // Create a top-level activity filter list
    activityFilterList = new ActivityRef.ActivityFilterList();

    // Create a create time filter
    activityFilterList.CreateTimeFilter = new ActivityRef.DateTimeFilter();

    // Create the date time criteria for this filter
    ActivityRef.CriteriaOr rootDtCriOr = new ActivityRef.CriteriaOr();
```

```
    ActivityRef.CriteriaAnd dtCriAnd1 = new ActivityRef.CriteriaAnd();
    ActivityRef.CriteriaAnd dtCriAnd2 = new ActivityRef.CriteriaAnd();
    ActivityRef.CriteriaDateTime dtCriAct1 = new ActivityRef.CriteriaDateTime();
    ActivityRef.CriteriaDateTime dtCriAct2 = new ActivityRef.CriteriaDateTime();
    ActivityRef.CriteriaDateTime dtCriAct3 = new ActivityRef.CriteriaDateTime();
    ActivityRef.CriteriaDateTime dtCriAct4 = new ActivityRef.CriteriaDateTime();

    // Assume we are in the same time zone as the server
    TimeZone localTimeZone = TimeZone.CurrentTimeZone;

    // Specify criteria 1 details
    dtCriAct1.FilterMode = ActivityRef.EDateTimeFilterModes.GreaterThan;

    // 8:00 AM 30/6/2012
    dtCriAct1.CriteriaValue = new DateTime(2012, 6, 30, 8, 0, 0, DateTimeKind.Local);

    // Get UTC offset in minutes to disambiguate the Date Time value
    dtCriAct1.OffsetMinutes =
        (short)localTimeZone.GetUtcOffset(dtCriAct1.CriteriaValue).TotalMinutes;

    // Specify criteria 2 details
    dtCriAct2.FilterMode = ActivityRef.EDateTimeFilterModes.LessThanOrEqual;

    // 9:00 AM 30/6/2012
    dtCriAct2.CriteriaValue = new DateTime(2012, 6, 30, 9, 0, 0, DateTimeKind.Local);
    dtCriAct2.OffsetMinutes =
        (short)localTimeZone.GetUtcOffset(dtCriAct2.CriteriaValue).TotalMinutes;

    // Specify criteria 3 details
    dtCriAct3.FilterMode = ActivityRef.EDateTimeFilterModes.GreaterThanOrEqual;

    // 4:00 PM 30/6/2012
    dtCriAct3.CriteriaValue = new DateTime(2012, 6, 30, 16, 0, 0, DateTimeKind.Local);

    // Get UTC offset in minutes to disambiguate the Date Time value
    dtCriAct3.OffsetMinutes =
        (short)localTimeZone.GetUtcOffset(dtCriAct3.CriteriaValue).TotalMinutes;

    // Specify criteria 4 details
    dtCriAct4.FilterMode = ActivityRef.EDateTimeFilterModes.LessThan;

    // 5:00 PM 30/6/2012
    dtCriAct4.CriteriaValue = new DateTime(2012, 6, 30, 17, 0, 0, DateTimeKind.Local);

    // Get UTC offset in minutes to disambiguate the Date Time value
    dtCriAct4.OffsetMinutes =
        (short)localTimeZone.GetUtcOffset(dtCriAct4.CriteriaValue).TotalMinutes;

    // Add the CriteriaActuals to the criteria ands
    dtCriAnd1.CriteriaList = new List<ActivityRef.Criteria>();
    dtCriAnd2.CriteriaList = new List<ActivityRef.Criteria>();
    dtCriAnd1.CriteriaList.Add(dtCriAct1);
    dtCriAnd1.CriteriaList.Add(dtCriAct2);
    dtCriAnd2.CriteriaList.Add(dtCriAct3);
    dtCriAnd2.CriteriaList.Add(dtCriAct4);

    // Add the CriteriaAnds to the root criteria or
    rootDtCriOr.CriteriaList = new List<ActivityRef.Criteria>();
    rootDtCriOr.CriteriaList.Add(dtCriAnd1);
    rootDtCriOr.CriteriaList.Add(dtCriAnd2);

    // Assign the root date time criteria to the date time filter
    activityFilterList.CreateTimeFilter.Criteria = rootDtCriOr;
}
```

Syntax C++

```
// Create a filter on create time using nested criteria
void createActivityFilter_CreateTimeNested(NativeFilter::ActivityFilterList*& pActivityFilterList)
{
    // Create a top-level activity filter list
    pActivityFilterList = new NativeFilter::ActivityFilterList();

    // Create a create time filter
    pActivityFilterList->CreateTimeFilter = new DateTimeFilter();

    // Create the string criteria for this filter
    CriteriaOr* pRootCriOr = new CriteriaOr();
    CriteriaAnd* pCriAnd1 = new CriteriaAnd();
    CriteriaAnd* pCriAnd2 = new CriteriaAnd();
    CriteriaDateTime* pCriDT1 = new CriteriaDateTime();
```

```
    CriteriaDateTime* pCriDT2 = new CriteriaDateTime();
    CriteriaDateTime* pCriDT3 = new CriteriaDateTime();
    CriteriaDateTime* pCriDT4 = new CriteriaDateTime();

    SYSTEMTIME st;
    ZeroMemory(&st, sizeof(SYSTEMTIME));

    // Specify criteria 1 details
    pCriDT1->FilterMode = EDateTimeFilterModes::GreaterThan;
    st.wYear = 2011;
    st.wMonth = 6;
    st.wDay = 30;
    st.wHour = 8;
    setDateTimeCriteriaValue(&st, pCriDT1);

    // Specify criteria 2 details
    pCriDT2->FilterMode = EDateTimeFilterModes::LessThanOrEqual;
    st.wHour = 9;
    setDateTimeCriteriaValue(&st, pCriDT2);

    // Specify criteria 3 details
    pCriDT3->FilterMode = EDateTimeFilterModes::GreaterThanOrEqual;
    st.wHour = 16;
    setDateTimeCriteriaValue(&st, pCriDT3);

    // Specify criteria 4 details
    pCriDT4->FilterMode = EDateTimeFilterModes::LessThan;
    st.wHour = 17;
    setDateTimeCriteriaValue(&st, pCriDT4);

    // Add the criteria actuals to the criteria and's
    pCriAnd1->CriteriaList.push_back(pCriDT1);
    pCriAnd1->CriteriaList.push_back(pCriDT2);
    pCriAnd2->CriteriaList.push_back(pCriDT3);
    pCriAnd2->CriteriaList.push_back(pCriDT4);

    // Add the criteria and's to the root criteria or
    pRootCriOr->CriteriaList.push_back(pCriAnd1);
    pRootCriOr->CriteriaList.push_back(pCriAnd2);

    // Assign the root date time criteria to the date time filter
    pActivityFilterList->CreateTimeFilter->Criteria = pRootCriOr;
}
```

Printing returned activities

Use the following function to print the returned activities.

Syntax C#

```
static void displayActivityList(ActivityRef.BatchInformationType bi)
{
    Console.WriteLine("Return Information:\n" +
        "\tCode: {0}, String: {1}\n",
        bi.ReturnInformation.ReturnCode,
        bi.ReturnInformation.ReturnString);

    int i = 1;
    foreach (ActivityRef.BatchListEntryType ble in
        bi.BatchList[0].BatchListEntry)
    {
        string outputFormat =
            "Activity {0}\n" +
            "    ID:                 {1}\n" +
            "    Description:        {2}\n" +
            "    BatchListEntryType:  {3}\n" +
            "        OtherValue:       {4}\n" +
            "    Status:             {5}\n" +
            "    Mode:               {6}\n" +
            "    RecipeID:           {7}\n" +
            "    BatchID:            {8}\n" +
            "    CreateTime:         {9}\n" +
            "    ActualStartTime:    {10}\n" +
            "    ActualEndTime:      {11}\n" +
            "    Cluster:            {12}\n" +
            "    Asset:              {13}\n" +
            "    PublicName:         {14}\n" +
            "    ModeAttribute:      {15}\n" +
```

```
            "    ExecutionStatus:     {16}\n" +
            "    ElementReturn:\n" +
            "        Code:              {17}\n" +
            "        String:            {18}\n\n";

        Console.WriteLine(
            outputFormat,
            i,
            ble.ID.Value,
            ble.Description[0].Value,
            ble.BatchListEntryType1.Value,
            ble.BatchListEntryType1.OtherValue,
            ble.Status.Value,
            ble.Mode.Value,
            ble.RecipeID.Value,
            ble.BatchID.Value,
            (ble.CreateTime == null) ?
                "None" : ble.CreateTime.Value.ToString("yyyy-MM-ddTHH:mm:sszzz"),
            (ble.ActualStartTime == null) ?
                "None" : ble.ActualStartTime.Value.ToString("yyyy-MM-ddTHH:mm:sszzz"),
            (ble.ActualEndTime == null) ?
                "None" : ble.ActualEndTime.Value.ToString("yyyy-MM-ddTHH:mm:sszzz"),
            ble.Cluster,
            ble.Asset,
            ble.PublicName,
            ble.ModeAttribute,
            ble.ExecutionStatus,
            ble.ElementReturn.ReturnCode,
            ble.ElementReturn.ReturnString);
        i++;
    }
}
```

Syntax C++

```
// Display a list of activities
void displayActivityList(BatchInformationType* batchInfo)
{
    for each (BatchListType* batchListType in batchInfo->BatchList)
    {
        for each (BatchListEntryType* batchListEntryType in batchListType->BatchListEntry)
        {
            wcout << "Activity" << endl;
            wcout << "   ID:                  "
                <<batchListEntryType->ID->Value << endl;
            wcout << "   Description:         "
                << ((DescriptionType*)batchListEntryType->Description.front())->Value << endl;
            wcout << "   BatchListEntryType:  "
                << batchListEntryType->BatchListEntryType1->Value << endl;
            wcout << "      OtherValue:       "
                << batchListEntryType->BatchListEntryType1->OtherValue << endl;
            wcout << "   Status:              "
                << ((BatchStatusType*)batchListEntryType->Status)->Value << endl;
            wcout << "   Mode:                "
                << ((ModeType*)batchListEntryType->Mode)->Value << endl;
            wcout << "   RecipeID:            "
                << ((RecipeIDType*)batchListEntryType->RecipeID)->Value << endl;
            wcout << "   BatchID:             "
                << ((BatchIDType*)batchListEntryType->BatchID)->Value << endl;
            wcout << "   CreateTime:          "
                << formatDateTimeString(batchListEntryType->CreateTime) << endl;
            wcout << "   ActualStartTime:     "
                << formatDateTimeString(batchListEntryType->ActualStartTime) << endl;
            wcout << "   ActualEndTime:       "
                << formatDateTimeString(batchListEntryType->ActualEndTime) << endl;
            wcout << "   Cluster:             "
                << batchListEntryType->Cluster << endl;
            wcout << "   Asset:               "
                << batchListEntryType->Asset << endl;
            wcout << "   PublicName:          "
                << batchListEntryType->PublicName << endl;
            wcout << "   ModeAttribute:       "
                << batchListEntryType->ModeAttribute << endl;
            wcout << "   ExecutionStatus:     "
                << batchListEntryType->ExecutionStatus << endl;
            wcout << "   ElementReturn:" << endl;
            wcout << "      Code:             "
                << batchListEntryType->ElementReturn->ReturnCode << endl;
            wcout << "      String:           "
                << batchListEntryType->ElementReturn->ReturnString << endl;
        }
```

```
    }
}

// Format a DateTime string including UTC offset information
wstring formatDateTimeString(DateTimeType* pDtt)
{
    // Check for NULL input    if (pDtt == NULL)
        return wstring(L"None");

    // Convert large integer to FILETIME
    FILETIME ft;
    ft.dwHighDateTime = ((LARGE_INTEGER*)&pDtt->Value)->HighPart;
    ft.dwLowDateTime = ((LARGE_INTEGER*)&pDtt->Value)->LowPart;

    // Pass onto master
    return formatDateTimeString(&ft, pDtt->OffsetMinutes);
}

// Format a DateTime string including UTC offset information
wstring formatDateTimeString(CriteriaDateTime* pDtCriAct)
{
    // Check for NULL input
    if (pDtCriAct == NULL)
        return wstring(L"None");

    // Convert large integer to FILETIME
    FILETIME ft;
    ft.dwHighDateTime = ((LARGE_INTEGER*)&pDtCriAct ->CriteriaValue)->HighPart;
    ft.dwLowDateTime = ((LARGE_INTEGER*)&pDtCriAct ->CriteriaValue)->LowPart;

    // Pass onto master
    return formatDateTimeString(&ft, pDtCriAct->OffsetMinutes);
}

// Format a DateTime string including UTC offset information
wstring formatDateTimeString(FILETIME* pDateTime, short offsetMins)
{
    // Convert FILETIME to SYSTEMTIME
    SYSTEMTIME st;
    FileTimeToSystemTime(pDateTime, &st);

    // Build a XML DateTime string
    // Format: YYYY-MM-DDTHH:mm:SSZZZ
    wstringstream wss;
    wss << st.wYear << L"-"
        << setw(2) << setfill(L'0') << st.wMonth << L"-"
        << setw(2) << setfill(L'0') << st.wDay << L"T"
        << setw(2) << setfill(L'0') << st.wHour << L":"
        << setw(2) << setfill(L'0') << st.wMinute << L":"
        << setw(2) << setfill(L'0') << st.wSecond;

    // Append timezone information
    if (offsetMins != 0)
    {
        // Assume the offset will always be in whole hour increments
        int offsetHours = 0;
        if (offsetMins < 0)
        {
            wss << L"-";
            offsetHours = (-1 * offsetMins) / 60;
        }
            else
            {
                wss << L"+";
                offsetHours = offsetMins / 60;
            }
            wss << setw(2) << setfill(L'0') << offsetHours
                << L":00";
        }
        else // UTC time
            wss << L"Z";
    // Return the formatted string
    return wss.str();
}
```

Activity entity filter: Type

The following filter restricts the returned activity entity list to only those activity entities that have an activity type of *Batch*.

Syntax C#

```
// Create a simple filter on activity entity type
static void CreateActivityEntityFilter_Type(
    ref ActivityRef.ActivityEntityFilterList
    activityEntityFilterList)
{
    // Create a top-level activity entity filter list
    pActivityEntityFilterList = new ActivityRef.ActivityEntityFilterList();

    // Create a new type filter
    pActivityEntityFilterList.TypeFilter = new ActivityRef.ActivityTypeFilter();

    // Specify the filtering mode
    pActivityEntityFilterList.TypeFilter.FilterMode = ActivityRef.EActTypeFilterModes.Equal;

    // Specify the activity type
    pActivityEntityFilterList.TypeFilter.ActivityType = ActivityRef.EActivityTypes.Batch;
}
```

Syntax C++

```
// Create a simple filter on activity entity type
void createActivityEntityFilter_Type(
    NativeFilter::ActivityEntityFilterList*&
    pActivityEntityFilterList)
{
    // Create a top-level activity entity filter list
    pActivityEntityFilterList = new NativeFilter::ActivityEntityFilterList();

    // Create a new type filter
    pActivityEntityFilterList->TypeFilter = new ActivityTypeFilter();

    // Specify the filtering mode
    pActivityEntityFilterList->TypeFilter->FilterMode = EActTypeFilterModes::Equal;

    // Specify the activity type
    pActivityEntityFilterList->TypeFilter->ActivityType = EActivityTypes::Batch;
}
```

Activity entity filter: Asset

The following filter restricts the returned activity entity list to only those activity entities that have a parent or ancestor asset with a tag name of *Building_XYZ*.

Syntax C#

```
// Create a simple filter on activity entity asset
static void CreateActivityEntityFilter_Asset(
    ref ActivityRef.ActivityEntityFilterList activityEntityFilterList)
{
    // Create a top-level activity entity filter list
    activityEntityFilterList = new ActivityRef.ActivityEntityFilterList();

    // Create a new asset filter
    activityEntityFilterList.AssetFilter = new ActivityRef.ActivityAssetFilter();

    // Specify which asset to filter on
    ActivityRef.CriteriaAsset asset = new ActivityRef.CriteriaAsset();
    asset.Asset = "Building_XYZ";
    pActivityEntityFilterList.AssetFilter.Criteria = asset;

}
```

Syntax C++

```
// Create a simple filter on activity entity asset
void createActivityEntityFilter_Asset(
    NativeFilter::ActivityEntityFilterList*& pActivityEntityFilterList)
{
    // Create a top-level activity entity filter list
    pActivityEntityFilterList = new NativeFilter::ActivityEntityFilterList();

    // Create a new asset filter
    pActivityEntityFilterList->AssetFilter = new ActivityAssetFilter();

    // Specify which asset to filter on
    CriteriaAsset* pCriAsset = new CriteriaAsset();
    pCriAsset->Asset = L"Building_XYZ";
    pActivityEntityFilterList->AssetFilter->Criteria = pCriAsset;
}
```

Activity entity filter: Server Base Name

The following filter restricts the returned activity entity list to only those activities that belong to the cluster *ExprR410Srv*. (The actual server names are *ExprR410SrvA* and *ExprR410SrvB*.)

Syntax C#

```
// Create a simple filter on cluster base name
static void CreateActivityEntityFilter_SrvBaseName(
    ref ActivityRef.ActivityEntityFilterList activityEntityFilterList)
{
    // Create a top-level activity entity filter list
    pActivityEntityFilterList = new ActivityRef.ActivityEntityFilterList();

    // Create a server base name filter
    pActivityEntityFilterList.ServerBaseNameFilter = new ActivityRef.StringFilter();

    // Create a string criteria to specify the server base name
    ActivityRef.CriteriaString criString = new ActivityRef.CriteriaString();

    // Specify the filter mode
    criString.FilterMode = ActivityRef.EStringFilterModes.Equal;

    // Specify the criteria value
    criString.CriteriaValue = "ExprR410Srv";

    // Assign the string criteria to the string filter
    pActivityEntityFilterList.ServerBaseNameFilter.Criteria = criString;
}
```

Syntax C++

```
// Create a simple filter on cluster base name
void createActivityEntityFilter_SrvBaseName(
    NativeFilter::ActivityEntityFilterList*& pActivityEntityFilterList)
{
    // Create a top-level activity entity filter list
    pActivityEntityFilterList = new NativeFilter::ActivityEntityFilterList();

    // Create a server base name filter
    pActivityEntityFilterList->ServerBaseNameFilter = new StringFilter();

    // Create a string criteria to specify the server base name
    CriteriaString* pCriStr = new CriteriaString();

    // Specify the filter mode
    pCriStr->FilterMode = EStringFilterModes::Equal;

    // Specify the criteria value
    pCriStr->CriteriaValue = L"ExprR410Srv";

    // Assign the string criteria to the string filter
    pActivityEntityFilterList->ServerBaseNameFilter->Criteria = pCriStr;
}
```

Activity entity filter: Public Name

The following filter restricts the returned activity entity list to only those activity entities that have a public name of *Resin recipe*.

Syntax C#

```
// Create a simple filter on public name
static void CreateActivityEntityFilter_PublicName(
    ref ActivityRef.ActivityEntityFilterList activityEntityFilterList)
{
    // Create a top-level activity filter list
    activityEntityFilterList = new ActivityRef.ActivityEntityFilterList();

    // Create a public name filter
    activityEntityFilterList.PublicNameFilter = new ActivityRef.StringFilter();

    // Create a string criteria to specify the public name
    ActivityRef.CriteriaString criString = new ActivityRef.CriteriaString();

    // Specify the filter mode
    criString.FilterMode = ActivityRef.EStringFilterModes.Equal;

    // Specify the criteria value
    criString.CriteriaValue = "Resin recipe";

    // Assign the string criteria to the string filter
    activityEntityFilterList.PublicNameFilter.Criteria = criString;
}
```

Syntax C++

```
// Create a simple filter on public name
void createActivityEntityFilter_PublicName(
    NativeFilter::ActivityEntityFilterList*& pActivityEntityFilterList)
{
    // Create a top-level activity filter list
    pActivityEntityFilterList = new NativeFilter::ActivityEntityFilterList();

    // Create a public name filter
    pActivityEntityFilterList->PublicNameFilter = new StringFilter();

    // Create a string criteria to specify the public name
    CriteriaString* pCriStr = new CriteriaString();

    // Specify the filter mode
    pCriStr->FilterMode = EStringFilterModes::Equal;

    // Specify the criteria value
    pCriStr->CriteriaValue = L"Resin recipe";

    // Assign the string criteria to the string filter
    pActivityEntityFilterList->PublicNameFilter->Criteria = pCriStr;
}
```

Activity entity filter: Point Build Date

The following filter restricts the returned activity entity list to only those activity entity's that have a point build date (*DateTime* when the activity entity was downloaded to the server) on or after 12:00 PM.

In this example the client and server are assumed to be in the same time zone.

Syntax C#

```
// Create a simple filter on point build date
static void createActivityEntityFilter_PntbldDate(
    out ActivityRef.ActivityEntityFilterList activityEntityFilterList)
{
    // Create a top-level activity entity filter list
    activityEntityFilterList = new ActivityRef.ActivityEntityFilterList();
```

```
    // Create an EndTime filter
    activityEntityFilterList.BuildDateFilter = new ActivityRef.DateTimeFilter();

    // Create a datetime criteria to specify the start time
    ActivityRef.CriteriaDateTime criDateTime = new ActivityRef.CriteriaDateTime();

    // Specify the filter mode    dateTimeCri.FilterMode =
        ActivityRef.EDateTimeFilterModes.GreaterThanOrEqual;

    // Assume we are in the same time zone as the server
    TimeZone localTimeZone = TimeZone.CurrentTimeZone;

    // Specify the criteria value
    // (Activity Entities downloaded on or after 12:00 PM today)
    DateTime n = DateTime.Now;    DateTime thisAfternoon =
        new DateTime(n.Year, n.Month, n.Day, hour: 12, minute: 0, second: 0, kind: n.Kind);
    criDateTime.CriteriaValue = thisAfternoon;

    // Get UTC offset in minutes
    criDateTime.OffsetMinutes =
        (short)localTimeZone.GetUtcOffset(criDateTime.CriteriaValue).TotalMinutes;

    // Assign the datetime criteria to the datetime filter
    activityEntityFilterList.BuildDateFilter.Criteria = criDateTime;
}
```

Syntax C++

```
// Create a simple filter on point build date
void createActivityEntityFilter_PntbldDate(
    NativeFilter::ActivityEntityFilterList*& pActivityEntityFilterList)
{
    // Create a top-level activity filter list
    pActivityEntityFilterList = new NativeFilter::ActivityEntityFilterList();

    // Create a build date filter
    pActivityEntityFilterList->BuildDateFilter = new DateTimeFilter();

    // Create a DateTime criteria to specify the build date
    CriteriaDateTime* pCriDT = new CriteriaDateTime();

    // Specify the filter mode
    pCriDT->FilterMode = EDateTimeFilterModes::GreaterThanOrEqual;

    // Get the local system time
    SYSTEMTIME st;
    GetLocalTime(&st);

    // Adjust to 12:00 PM
    st.wHour = 12;
    st.wMinute = 0;
    st.wSecond = 0;
    st.wMilliseconds = 0;

    // Specify the criteria value
    setDateTimeCriteriaValue(&st, pCriDT);

    // Assign the DateTime criteria to the DateTime filter
    pActivityEntityFilterList->BuildDateFilter->Criteria = pCriDT;
}
```

Printing returned activity entities

Use the following function to print the returned activity entities.

Syntax C#

```
// Display a list of activity entities
static void DisplayActivityEntityList(ActivityRef.BatchInformationType bi)
{
    Console.WriteLine(
        "Return Information:\n\tCode: {0}, String: {1}\n",
        bi.ReturnInformation.ReturnCode,
        bi.ReturnInformation.ReturnString);

    int i = 1;
    foreach (ActivityRef.MasterRecipeType mr in bi.MasterRecipe)
```

```
    {
        string outputFormat =
            "Activity {0}\n" +
            "    ID:                    {1}\n" +
            "    Description:           {2}\n" +
            "    Cluster:               {3}\n" +
            "    Asset:                 {4}\n" +
            "    ActivityType:          {5}\n" +
            "    PublicName:            {6}\n" +
            "    PntbldDate:            {7}\n" +
            "    ElementReturn:\n" +
            "        Code:                {8}\n" +
            "        String:              {9}\n\n";

        Console.WriteLine(
            outputFormat,
            i,
            mr.ID.Value,
            mr.Description[0].Value,
            mr.Cluster,
            mr.Asset,
            mr.ActivityType.Value,
            mr.PublicName,
            (mr.PntbldDate == null) ?
                "None" : mr.PntbldDate.Value.ToString("yyyy-MM-ddTHH:mm:sszzz"),
            mr.ElementReturn.ReturnCode,
            mr.ElementReturn.ReturnString);
        i++;
    }
}
```

Syntax C++

```
// Display a list of activity entities
void displayActivityEntityList(BatchInformationType* batchInfo)
{
    for each (MasterRecipeType* masterRecipeType in
        batchInfo->MasterRecipe)
    {
        wcout << "Activity Entity" << endl;
        wcout << "   ID:                  "
            << masterRecipeType->ID->Value << endl;
        wcout << "   Description:          "
            << ((DescriptionType*)masterRecipeType
                ->Description.front())->Value << endl;
        wcout << "   Cluster:              "
            << masterRecipeType->Cluster << endl;
        wcout << "   Asset:                "
            << masterRecipeType->Asset << endl;
        wcout << "   ActivityType:        "
            << masterRecipeType->ActivityType->Value <<endl;
        wcout << "   PublicName:           "
            << masterRecipeType->PublicName << endl;
        wcout << "   PntbldDate:           "
            << formatDateTimeString(masterRecipeType->PntbldDate) << endl;
    }
}

// Format a DateTime string including UTC offset information
wstring formatDateTimeString(DateTimeType* pDtt)
{
    // Check for NULL input
    if (pDtt == NULL)
        return wstring(L"None");

    // Convert large integer to FILETIME
    FILETIME ft;
    ft.dwHighDateTime = ((LARGE_INTEGER*)&pDtt->Value)->HighPart;
    ft.dwLowDateTime = ((LARGE_INTEGER*)&pDtt->Value)->LowPart;

    // Pass onto master
    return formatDateTimeString(&ft, pDtt->OffsetMinutes);
}

// Format a DateTime string including UTC offset information
wstring formatDateTimeString(CriteriaDateTime* pDtCriAct)
{
    // Check for NULL input
    if (pDtCriAct == NULL)
        return wstring(L"None");
```

```
    // Convert large integer to FILETIME
    FILETIME ft;
    ft.dwHighDateTime = ((LARGE_INTEGER*)&pDtCriAct->CriteriaValue)->HighPart;
    ft.dwLowDateTime = ((LARGE_INTEGER*)&pDtCriAct->CriteriaValue)->LowPart;

    // Pass onto master
    return formatDateTimeString(&ft, pDtCriAct->OffsetMinutes);
}

// Format a DateTime string including UTC offset information
wstring formatDateTimeString(FILETIME* pDateTime, short offsetMins)
{
    // Convert FILETIME to SYSTEMTIME
    SYSTEMTIME st;
    FileTimeToSystemTime(pDateTime, &st);

    // Build a XML DateTime string
    // Format: YYYY-MM-DDTHH:mm:SSZZZ
    wstringstream wss;
    wss << st.wYear << L"-"
        << setw(2) << setfill(L'0') << st.wMonth << L"-"
        << setw(2) << setfill(L'0') << st.wDay << L"T"
        << setw(2) << setfill(L'0') << st.wHour << L":"
        << setw(2) << setfill(L'0') << st.wMinute << L":"
        << setw(2) << setfill(L'0') << st.wSecond;

    // Append timezone information
    if (offsetMins != 0)
    {

        // Assume the offset will always be in whole hour increments
        int offsetHours = 0;
        if (offsetMins < 0)
        {
            wss << L"-";
            offsetHours = (-1 * offsetMins) / 60;
        }
        else
        {
            wss << L"+";
            offsetHours = offsetMins / 60;
        }
        wss << setw(2) << setfill(L'0') << offsetHours
            << L":00";
    }
    else // UTC time
        wss << L"Z";

    // Return the formatted string
    return wss.str();
}
```

Error scenarios and exception handling

The following sections describe how to handle common error scenarios and manage redundancy.

Related topics

"Faulted proxy" on page 435
"Handling Partial Function Fail" on page 435

Faulted proxy

The following code demonstrates how a faulted proxy might be checked and recovered.

Syntax C#

```
static void ExecuteOperation(ref ActivityRef.AppServicesClient proxy)
{
    // Check if our proxy is faulted
    if (proxy.State == System.ServiceModel.CommunicationState.Faulted)
    {
        // Our proxy is faulted so create a new one
        proxy = new ActivityRef.AppServicesClient();

}

    // Try to use the API and catch any exceptions
    try
    {
        // This operation could fail, throwing an exception
        proxy.IsPrimary();
    }
    catch (Exception ex) // Catch any exception
    {
        // Our proxy could now be faulted
        // We could also query the proxy's state and recover here
    }
}
```

Syntax C++

```
static void ExecuteOperation(AppServicesAPI* proxy)
{
    // Check if our proxy is faulted
    if (proxy->GetState() == ProxyState::FAULTED)
    {
        proxy = new AppServicesAPI(L"as01hsc410svra");
    }
    // Try to use the API and catch any exceptions
    try
    {
        // This operation could fail, throwing an exception
        proxy->IsPrimary();
    }
    catch (exception ex) // Catch any exception
    {
        // Our proxy could now be faulted
        // We could also query the proxy's state and recover here
    }
}
```

Handling Partial Function Fail

An error code you can expect to encounter is *Partial Function Fail* (Hex: `0xC8C7`, Dec: `51399`). This code indicates that one or many element(s) involved in an operation has failed and as such the operation has partially failed.

The following code demonstrates reading the value of three point parameters, one which is invalid. The read for the point parameter that is invalid will fail but the others will succeed. In this example the list of return values is traversed to identify which elements failed.

Attention

This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C#

```
// Create a Point Parameter object for reading the Activity's parameter values
ActivityRef.PointParamType pointParam = new ActivityRef.PointParamType();
pointParam.PointList = new List<ActivityRef.PointType>(1);

// Create a Point object
ActivityRef.PointType point = new ActivityRef.PointType();
point.Parameter = new List<ActivityRef.BatchParameterType>();
point.ID = activity.ID.Value; // Provide the ID of the activity
pointParam.PointList.Add(point);

ActivityRef.BatchParameterType param = null;

param = new ActivityRef.BatchParameterType();
param.ID = new ActivityRef.IDType();
param.ID.Value = "actstate";
point.Parameter.Add(param);

param = new ActivityRef.BatchParameterType();
param.ID = new ActivityRef.IDType();
param.ID.Value = "execsts";
point.Parameter.Add(param);

param = new ActivityRef.BatchParameterType();
param.ID = new ActivityRef.IDType();
param.ID.Value = "nonExistentParameter";
point.Parameter.Add(param);

/////////////////////////////////////////////////////
// Read the parameter values
int opResult = 0;
opResult = proxy.GetPointParameterValues(
    ref pointParam, (int)ActivityRef.ESubscriptionPeriod.CACHE_READ);

// Check for and handle a partial function failure
List<ActivityRef.BatchParameterType> failedReadsList = null;
if (opResult == (int)ActivityRef.EReturnCode.PARTIAL_FUNC_FAIL)
{
    // List for recording failed reads
    failedReadsList = new List<ActivityRef.BatchParameterType>();

    // Find the element[s] that failed
    foreach (ActivityRef.BatchParameterType p in pointParam.PointList[0].Parameter)
    {
        // Element return code is a string so convert it to an int
        // Converting from hex so need to work in base 16
        int elemCode = Convert.ToInt32(p.ElementReturn.ReturnCode, 16);
        if (elemCode != (int)ActivityRef.EReturnCode.SUCCESS) failedReadsList.Add(p);
    }
}

// failedReadsList contains any failed parameter reads
// It can be used to exclude these from further processing
// or reported to the user, etc
```

Syntax C++

```
// Create a Point Parameter object for reading the Activity's parameter values
PointParamType* pointParam = new PointParamType();

// Create a Point object
PointType* point = new PointType();

// Provide the ID of the activity
point->ID = activity->ID->Value;
pointParam->PointList.push_back(point);

BatchParameterType* param = NULL;
```

```
param = new BatchParameterType();
param->ID = new IDType();
param->ID->Value = L"actstate";
point->Parameter.push_back(param);

param = new BatchParameterType();
param->ID = new IDType();
param->ID->Value = L"execsts";
point->Parameter.push_back(param);

param = new BatchParameterType();
param->ID = new IDType();
param->ID->Value = L"nonExistentParameter";
point->Parameter.push_back(param);

////////////////////////////////////////////////////////
// Read the parameter values
int opResult = 0;
opResult = proxy->GetPointParameterValues(
    pointParam, AppServicesDefs::SUBSCRIPTION_CACHE_READ);

// Check for and handle a partial function failure
list<BatchParameterType*> failedReadsList;
if (opResult == AppServicesDefs::PARTIAL_FUNC_FAIL)
{
    // Find the element[s] that failed
    for each (BatchParameterType* p in pointParam->PointList.front()->Parameter)
    {
        // Element return code is a string so convert it to an int
        // Converting from hex so need to work in base 16
        int elemCode = wcstol(p->ElementReturn->ReturnCode.c_str(), NULL, 16);

        if (elemCode != AppServicesDefs::SUCCESS)
            failedReadsList.push_back(p);
    }
}
```

Connecting to multiple servers

Multiple servers can be connected to simultaneously through the use of multiple proxy objects. The following code demonstrates constructing three proxy objects to three different servers; a redundant server pair, and a standalone server.

Attention

This code should be enclosed in a try catch block with appropriate exception handling. For more information about exception handling, see “Error scenarios and exception handling” on page 435.

Syntax C#

```
ActivityRef.AppServicesClient standAloneSrvProxy =
    new ActivityRef.AppServicesClient("NetTcpEndpoint_IActivity",
    "net.tcp://standAloneSrv:40200/Services/" +
    "AppServices/Activity");

ActivityRef.AppServicesClient redundantSrvAProxy =
    new ActivityRef.AppServicesClient("NetTcpEndpoint_IActivity",
    "net.tcp://redundantSrvA:40200/Services/" +
    "AppServices/AActivity");

ActivityRef.AppServicesClient redundantSrvBProxy =
    new ActivityRef.AppServicesClient("NetTcpEndpoint_IActivity",
    "net.tcp://redundantSrvB:40200/Services/" +
    "AppServices/AActivity");

System.Console.WriteLine(
    "standAloneSrv is primary ? {0}",
    standAloneSrvProxy.IsPrimary());
System.Console.WriteLine(
    "redundantSrvA is primary ? {0}",
    redundantSrvAProxy.IsPrimary());
System.Console.WriteLine(
    "redundantSrvB is primary ? {0}",
    redundantSrvBProxy.IsPrimary());
```

This code sample produces the following output.

```
standAloneSrv is primary ? True
redundantSrvA is primary ? True
redundantSrvB is primary ? False
```

Syntax C++

```
AppServicesAPI* standAloneSrvProxy =
    new AppServicesAPI(L"net.tcp://standAloneSrv:40200/"
    "Services/AppServices/Activity");
AppServicesAPI* redundantSrvAProxy =
    new AppServicesAPI(L"net.tcp://redundantSrvA:40200/"
    "Services/AppServices/Activity");
AppServicesAPI* redundantSrvBProxy =
    new AppServicesAPI(L"net.tcp://redundantSrvB:40200/"
    "Services/AppServices/Activity");

cout << "standAloneSrv is primary ? "
    << standAloneSrvProxy->IsPrimary();
cout << "redundantSrvA is primary ? "
    << redundantSrvAProxy->IsPrimary();
cout << "redundantSrvB is primary ? "
    << redundantSrvBProxy->IsPrimary();
```

This code sample produces the following output.

```
standAloneSrv is primary ? True
redundantSrvA is primary ? True
redundantSrvB is primary ? False
```

Managing redundancy

In this section of the tutorial, you will complete the following tasks:

Task	Go to…
Install the Batch Application Services Client Component	"Installing the Batch Application Services Client Component" on page 440
Configure the Batch Application Services Client Component	"Configuring the Batch Application Services Client Component" on page 443
Configure your client application to redirect messages to the redundancy service	"Configuring your client application to redirect messages to the redundancy service" on page 444
Configure client proxies to communicate with redundant servers	"Using client proxies to communicate with redundant servers" on page 445

Installing the Batch Application Services Client Component

The Batch Application Services Client Component helps client applications to communicate to Experion redundant servers without having to know which server is primary.

To install the Batch Application Services Client Component

1. The Batch Application Services Client Component is installed from the Experion Media.
2. From the **Setup type of Node to install** dialog, click **Optional Features**, and then click **Next**.

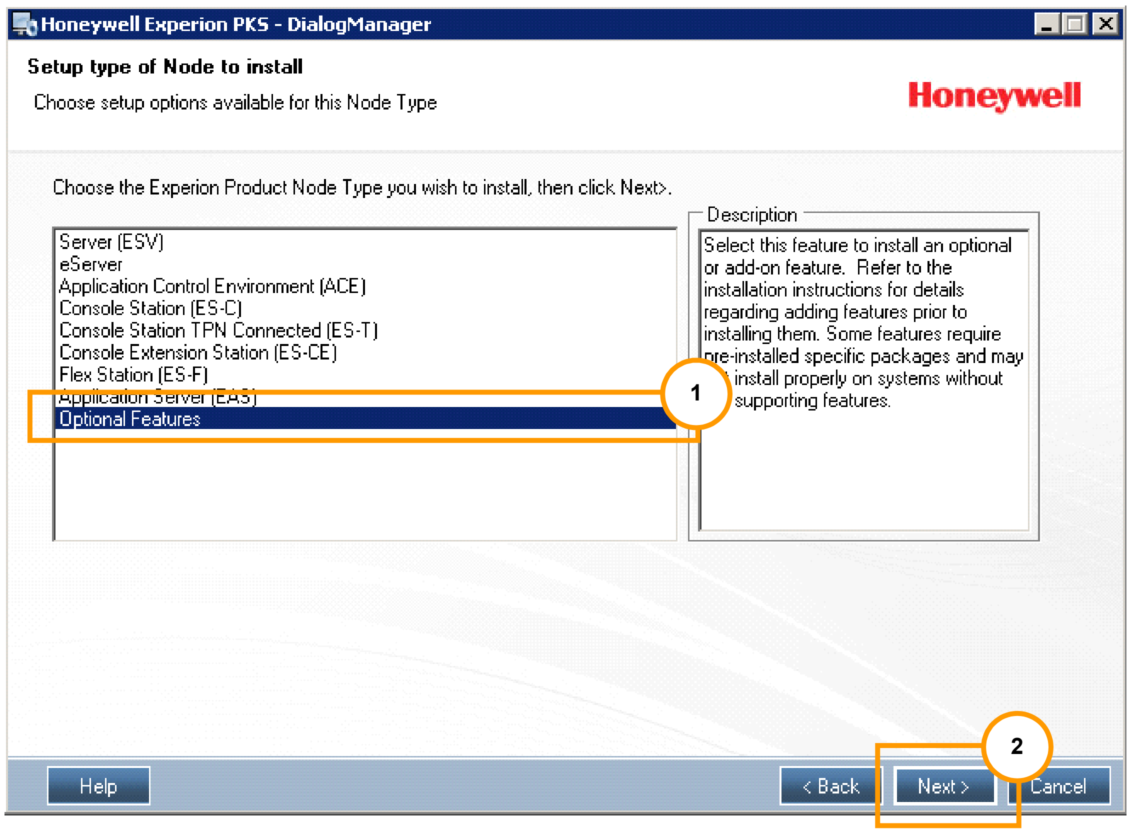


Figure 53: Optional Features

Item	Description
1	Optional Features selection.
2	Next button.

3. From the **Feature and Options Selection** dialog, click **Batch Application Services Client Component**, and then click **Next**.

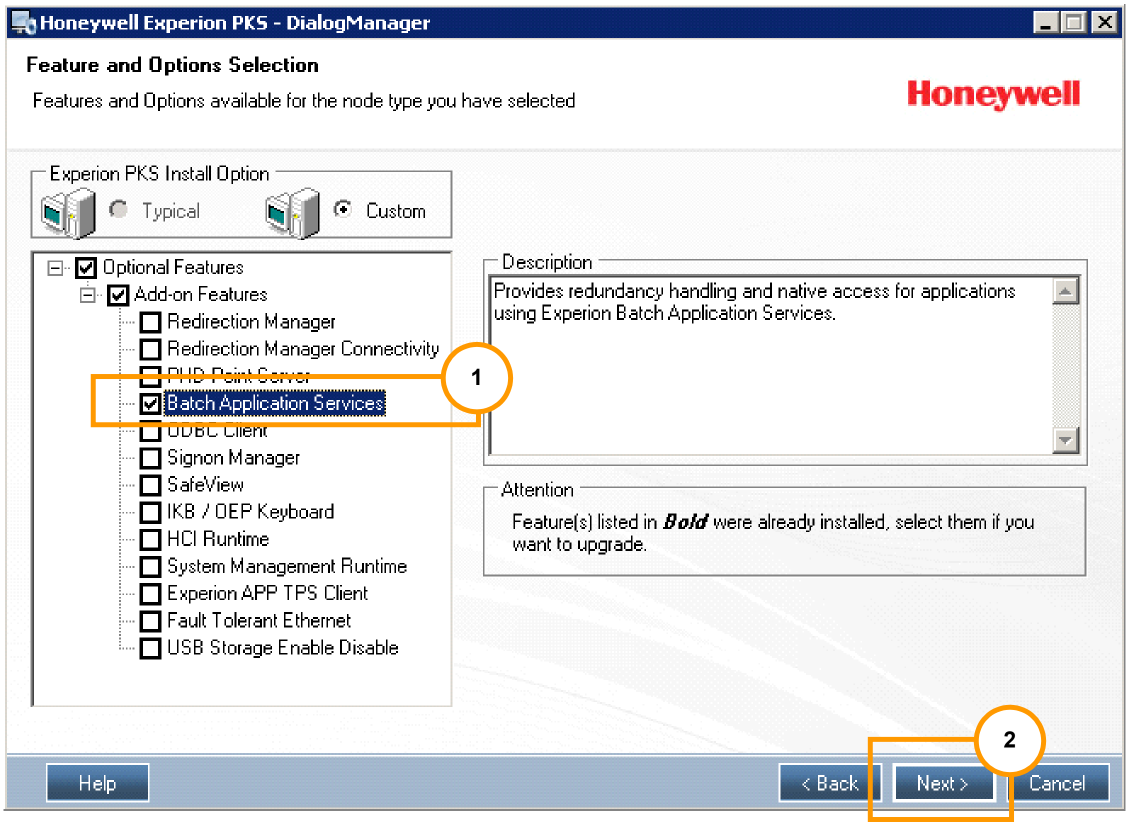


Figure 54: Feature and Options Selection

Item	Description
1	Batch Application Services selection.
2	Next button.

4 At the Summary page, click **Install**.
The installation process begins.

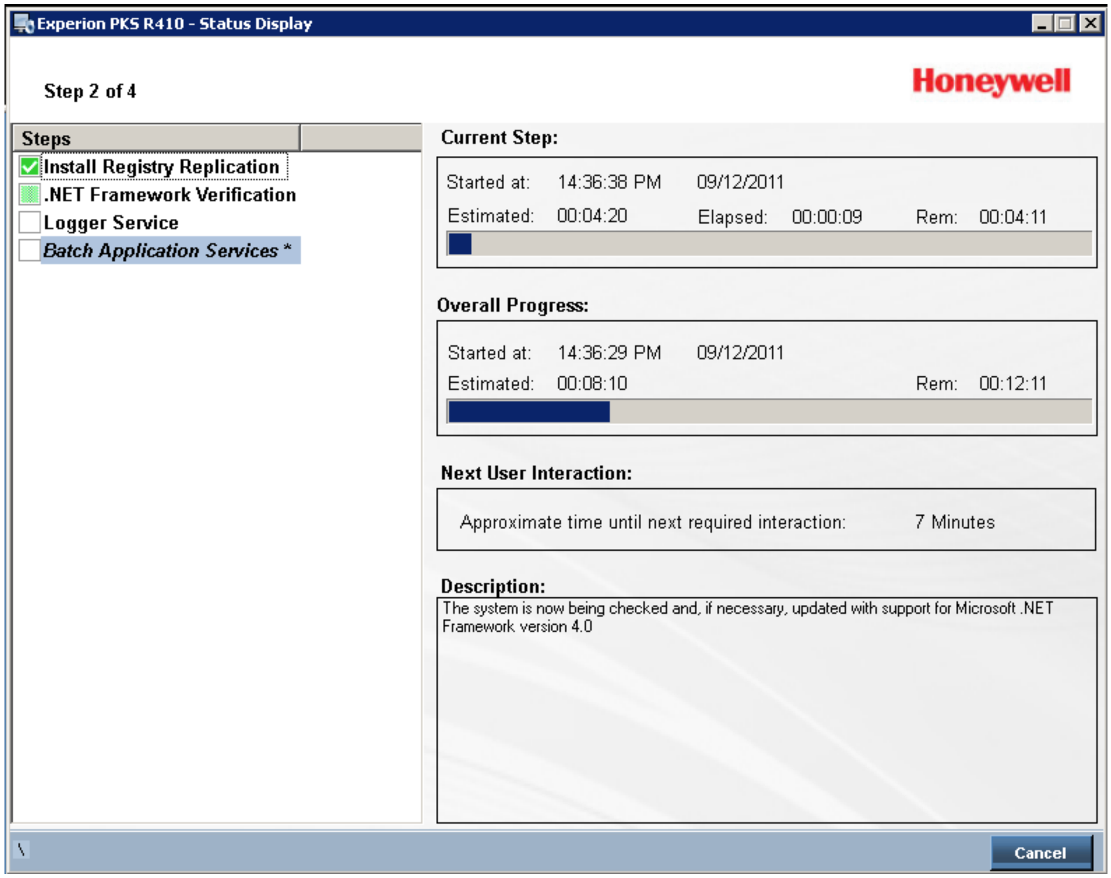


Figure 55: Installation progress

During the installation, the Experion Logger service will be installed if it is not already present.

Once installation is complete, the Experion PKS Batch Application Services Client Component will be listed in Windows Services.

Configuring the Batch Application Services Client Component

Once you have installed the Batch Application Services Client Component, you will configure it so that it contains all the servers to which your client application will connect. These server connections are defined as *endpoints*, and an endpoint should exist for each server listed in the hosts file.

For more information about editing the hosts file, see 'Editing the host file' in the *Server and Client Configuration Guide* .

By default, the Batch Application Services Client Component is hosted by the executable: *C:\Program Files(x86)\Honeywell\Client\RedundancyService\Honeywell.Client.RedundancyService.exe*.

The associated configuration file named *Honeywell.Client.RedundancyService.exe.config* contains all of the server endpoints.

```
<client>
<!-- Add your endpoints here. Note there is one endpoint
    per server connection. The redundancy service relies
    on the Experion server naming convention to identify
    redundant pairs -->
<endpoint
    address= "net.tcp://testsvra:40200/Services/AppServices/Activity/"
    binding="netTcpBinding"
    bindingConfiguration="RouterNetTCPBinding"
    behaviorConfiguration= "Honeywell.Client.Application.LargeListBehavior"
    contract= "Honeywell.Client.Services.RedundancyService.IRoutService"
    name="TESTSVRA" />
<endpoint
    address= "net.tcp://testsvrb:40200/Services/AppServices/Activity/"
    binding="netTcpBinding"
    bindingConfiguration="RouterNetTCPBinding"
    behaviorConfiguration=   "Honeywell.Client.Application.LargeListBehavior"
    contract= "Honeywell.Client.Services.RedundancyService.IRoutService"
    name="TESTSVRB" />
</client
```

To configure the Batch Application Services Client Component

- Add an endpoint for each server connection. Copy and paste one of the example endpoints and change the host name and name. The host name for each of these endpoints is the same as those defined in the hosts file.
  - **a** For FTE redundant servers, add one endpoint for each of the A and B servers.
  - **b** For Redundant Dual Network servers, add one endpoint for each of the A0, A1, B0, and B1 link connections.
  - **c** For Single Dual Network servers, add one endpoint for each of the 0 and 1 link connections.

Configuring your client application to redirect messages to the redundancy service

Client applications can use the *WCF ClientVia* behavior to redirect messages to the redundancy service.

- For C# applications, add behavior to the client application's configuration file.
- For C++ applications, add behavior to the *Honeywell.Client.AppServicesProxy.dll* configuration file.

To configure your client application to redirect messages to the redundancy service

1 In the appropriate configuration file, define the behavior as follows.

```
<behaviors>
    <endpointBehaviors>
        <behavior name= "Honeywell.Client.Application.Services.ActivityBehavior">
            <dataContractSerializer maxItemsInObjectGraph="1000000"/>
            <clientVia viaUri="net.tcp://localhost:40100/routingservice/router" />
        </behavior>
    </endpointBehaviors>
</behaviors>
```

2 Modify the client endpoint so that it uses this behavior.

```
<client>
    <endpoint
        address= "net.tcp://localhost:40200/Services/AppServices/Activity"
        behaviorConfiguration= "Honeywell.Client.Application.Services.ActivityBehavior"
        binding="netTcpBinding"
        bindingConfiguration="NetTcpEndpoint_IActivity"
        contract="AppServicesRef.AppServices"
        name="NetTcpEndpoint_IActivity">
    </endpoint>
</client>
```

Using client proxies to communicate with redundant servers

When creating a proxy object in your client application, you can use the host name, the IP address, or the base name as part of the endpoint address.

- When using the host name or IP address, all messages will be routed to this destination.
- When using the base name, all messages will be routed to the primary server of the redundant pair.

The Batch Application Services Client Component uses impersonation to ensure that the connection to the server is authenticated against the client application user.

To allow impersonation for redundancy in C# applications

- For C# applications, the proxy created on the client needs to be modified to allow impersonation. The following code snippet details how to allow impersonation for redundancy.

```
AppServicesClient proxyA =
    new AppServicesClient("NetTcpEndpoint_IActivity",
    "net.tcp://testsvra:40200/Services/AppServices/Activity/");

// Adding allow impersonation for redundancy
proxyA.ClientCredentials.Windows.AllowedImpersonationLevel =
    System.Security.Principal.TokenImpersonationLevel.Impersonation;
```

Attention

C++ applications do not need to complete this step because impersonation for redundancy is already pre-configured.

To configure C# application client proxies to communicate with redundant servers

- The following code snippets show how to configure C# application client proxies to communicate to redundant servers.

```
AppServicesClient proxyA =
    new AppServicesClient("NetTcpEndpoint_IActivity",
    "net.tcp://testsvra:40200/Services/AppServices/Activity/");

// Adding allow impersonation for redundancy
proxyA.ClientCredentials.Windows.AllowedImpersonationLevel =

    System.Security.Principal.TokenImpersonationLevel.Impersonation;

AppServicesClient proxyB =
    new AppServicesClient("NetTcpEndpoint_IActivity",
    "net.tcp://testsvrb:40200/Services/AppServices/Activity/");

// Adding allow impersonation for redundancy
proxyB.ClientCredentials.Windows.AllowedImpersonationLevel =

    System.Security.Principal.TokenImpersonationLevel.Impersonation;

AppServicesClient proxyBase =
    new AppServicesClient("NetTcpEndpoint_IActivity",
    "net.tcp://testsvr:40200/Services/AppServices/Activity/");

// Adding allow impersonation for redundancy
proxyBase.ClientCredentials.Windows.AllowedImpersonationLevel =

    System.Security.Principal.TokenImpersonationLevel.Impersonation;
```

- *proxyA* is a proxy to the A Server.
- *proxyb* is a proxy to the B Server.
- *proxyBase* is a proxy to the server pair, which will be automatically routed to the primary server.

To configure C++ application client proxies to communicate with redundant servers

- The following code snippets show how to configure C++ application client proxies to communicate to redundant servers.

```
// Create a proxy to communicate with Batch Application Services
AppServicesAPI proxyA = AppServicesAPI(L"testsvra");
AppServicesAPI proxyB = AppServicesAPI(L"testsvrb");
AppServicesAPI proxyBase = AppServicesAPI(L"testsvr");
```

  - *proxyA* is a proxy to the A Server.
  - *proxyb* is a proxy to the B Server.
  - *proxyBase* is a proxy to the server pair, which will be automatically routed to the primary server.

Using Class Based Recipes

A class based recipe (or CBR) can have zero, one, or more *unit classes* or *units* associated with it, each representing a collection of equipment. A *unit class* is a user-defined template that provides structure for a *unit* and the structure for building and executing the CBR. You can instantiate multiple *units* from a *unit class*, giving a CBR a choice of resources. Specify the *unit* to use for a *unit class* after a CBR activity has been created but before it is started.

Getting information about a CBR's units

For a CBR, *GetActivityEntityMetadata()* returns additional information about the *unit classes* and *units* associated with it. The following code snippets show how this information can be extracted.

Syntax C#

```
ActivityRef.BatchInformationType batchInfoContainingMetadata =
    new ActivityRef.BatchInformationType();

// ... GetActivityEntityMetadata code omitted

// accessing parameter details
foreach (ActivityRef.EnumerationSetType unitClass in
    batchInfoContainingMetadata.EnumerationSet)
{
    Console.WriteLine("Unit Class Name: ", unitClass.ID);
    foreach (ActivityRef.EnumerationType unit in unitClass.Enumeration)
    {
        Console.WriteLine("\tUnit: " + unit.EnumerationString);
    }
}
```

Syntax C++

```
BatchInformationType* batchInfoContainingMetadata =
    new BatchInformationType();

// ... GetActivityEntityMetadata code omitted

// accessing parameter details
for each (EnumerationSetType* unitClass in
    batchInfoContainingMetadata->EnumerationSet)
{
    wcout << "Unit Class Name: " << unitClass->ID;
    for each (EnumerationType* unit in unitClass->Enumeration)
    {
        wcout << "\tUnit: " << unit->EnumerationString;
    }
}
```

Unit parameters in a CBR can be identified by checking for a valid *EnumerationSetID* field. The *IsPrimary* field in a *unit* parameter indicates whether the *unit* is primary.

Syntax C#

```
ActivityRef.MasterRecipeType masterRecipeContainingMetadata =
    batchInfoContainingMetadata.MasterRecipe[0];

// Extract the details of the metadata returned
foreach (ActivityRef.BatchParameterType param in
    masterRecipeContainingMetadata.Formula.Parameter)
{
    // ...

    // Extract the details of each parameter value
    foreach (ActivityRef.BatchValueType value in param.Value)
    {
        if (param.IsPrimaryUnitSpecified)
        {
            Console.WriteLine("Unit is Primary: {0}",
                param.IsPrimaryUnit);
```

```
        }
        foreach (ActivityRef.BatchValueType value in param.Value)
        {
            if (value.EnumerationSetID != null && value.EnumerationSetID[0] != null)
            {
                Console.WriteLine("Unit Class Name: {0}", value.EnumerationSetID[0].Value);
            }
        }
    }
}
```

Syntax C++

```
MasterRecipeType* masterRecipeContainingMetadata =
    batchInfoContainingMetadata->MasterRecipe.front();

// Extract the details of the metadata returned
for each (BatchParameterType* param in
    masterRecipeContainingMetadata->Formula->Parameter)
{
     // Extract the details of each parameter value
    for each (BatchValueType* value in param->Value)
    {
        if (param->IsPrimaryUnitSpecified)
        {
            wcout << "Unit is Primary: " << param->IsPrimaryUnit;
        }
        for each (BatchValueType* value in param->Value)
        {
            if (value->EnumerationSetID.front() != NULL)
            {
                wcout << "Unit Class Name: " << value->EnumerationSetID.front()->Value;
            }
        }
    }
}
```

Selecting a unit for a CBR activity

Having collected information about *units* of a CBR activity, we can now use *SetPointParameterValues()* to select the *unit* for each unit parameter.

Suppose the following information were collected:

```
Activity ID: SERVER_559:$ACTIVITY_E303E8F6F8A13A
    Unit Parameter ID: Boiler
    Unit Class Name: BoilerSet
    Unit Parameter ID: Cooler
    Unit Class Name: CoolerSet
    Unit Class Name: BoilerSet
    Unit: Boiler1
    Unit: Boiler2
    Unit Class Name: CoolerSet
    Unit: Cooler1
    Unit: Cooler2
```

The snippets below demonstrate how to set *BoilerSet* to *Boiler1* and *CoolerSet* to *Cooler2*.

Syntax C#

```
ActivityRef.PointParamType pointParam =
    new ActivityRef.PointParamType();
pointParam.PointList = new List<ActivityRef.PointType>(1);
ActivityRef.PointType point = new ActivityRef.PointType();
point.Parameter = new List<ActivityRef.BatchParameterType>(2);

// Provide the ID of the activity
point.ID = "SERVER_559:$ACTIVITY_E303E8F6F8A13A";

// configuring the first unit parameter
ActivityRef.BatchParameterType param1 = new ActivityRef.BatchParameterType();

// ...
param1.ID.Value = "BoilerSet";
param1.Value[0].ValueString[0].Value = "Boiler1";

// configuring the second unit parameter
```

```
ActivityRef.BatchParameterType param2 = new ActivityRef.BatchParameterType();

// ...
param2.ID.Value = "CoolerSet";
param2.Value[0].ValueString[0].Value = "Cooler2";

point.Parameter.Add(param1);
point.Parameter.Add(param2);
pointParam.PointList.Add(point);

// Set the new values of the Activity's parameters
int opResult = 0;
opResult = proxy.SetPointParameterValues(ref pointParam);
```

Syntax C++

```
PointParamType* pointParam = new PointParamType();
PointType* point = new PointType();

// Provide the ID of the activity
point->ID = L"SERVER_559:$ACTIVITY_E303E8F6F8A13A";

// configuring the first unit parameter
BatchParameterType* param1 = new BatchParameterType();
// ...
param1->ID->Value = L"BoilerSet";
param1->Value[0]->ValueString[0]->Value = L"Boiler1";

// configuring the second unit paramter
BatchParameterType* param2 = new BatchParameterType();
// ...
param2->ID->Value = L"CoolerSet";
param2->Value[0]->ValueString[0]->Value = L"Cooler2";

point->Parameter.push_back(param1);
point->Parameter.push_back(param2);
pointParam->PointList.push_back(point);

// Set the new values of the Activity's parameters
int opResult = 0;
opResult = proxy->SetPointParameterValues(pointParam);
```

Batch Application Services reference

The Batch Application Services reference describes the BatchML schema.

This section also describes BatchML schema extensions, which are custom extensions specific to Experion. Also included in this section are class diagrams for the BatchML Object Model.

Related topics

"BatchML schema" on page 452
"BatchML schema extensions" on page 461
"BatchML Object Model class diagrams" on page 463

BatchML schema

This section describes the subset of elements in the BatchML schema that Experion supports. This section also describes custom extended elements, specific to Experion.

Attention
Those BatchML elements that not supported by Experion are not included in this document.

Related topics

"BatchInformation schema" on page 452
"MasterRecipe Type schema" on page 453
"ControlRecipe Type schema" on page 454
"BatchList schema" on page 456
"BatchListEntry schema" on page 456
"Formula Element schema" on page 458
"Parameter Element schema" on page 458
"Value Element schema" on page 459

BatchInformation schema

The *BatchInformation* element serves as a container. It may be made up of zero or more top-level elements.

MasterRecipe

A *MasterRecipe* is a template recipe that is used to create control recipes and *BatchListEntry* strings.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	See "MasterRecipe Type schema" on page 453.

ControlRecipe

A *ControlRecipe* is the same format as a master recipe, with additional information about execution.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	See "ControlRecipe Type schema" on page 454.

BatchList

A *BatchList* contains the list of batches for a process cell. In Experion, it is used to represent one or more Activities.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	See "BatchList schema" on page 456.

Version

Experion *Version* for extensions made to BatchML structures.

Attention
Version is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	1

ReturnInformation

`ReturnInformation` is the overall call return.

> **Attention**
> `ReturnInformation` is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
Structure	0 or 1	See “ReturnInformation schema” on page 461.

MasterRecipe Type schema

The `MasterRecipe` is a template recipe that is used to create control recipes and `BatchListEntry` strings.

ID

A string containing an identification of an element. In Experion, Activity Entity prepended tag name.

Type	# of elements expected	Example/Notes
String	1 only	`Server1:RCM_MakeResin`

Description

A string containing a description of an element. In Experion, the description of Activity Entity.

Type	# of elements expected	Example/Notes
String	0 to unbounded	`Super strong polymer blend`

Formula

A recipe's formula is a category of information that includes process inputs, process parameters, and process outputs.

Type	# of elements expected	Example/Notes
Structure	0 or 1	See “Formula Element schema” on page 458.

Cluster

Server base name where Activity Entity defined.

> **Attention**
> `Cluster` is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	`Server1`

Asset

Activity Entity asset assignment.

Attention
Asset is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	*XYZ*

ActivityType

Activity Entity Type (*Batch* or *Procedure*).

Attention
ActivityType is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
Enumeration	0 or 1	*Batch*

PublicName

Public name associated with the Activity entity.

Attention
PublicName is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	*ResinRecipe*

ElementReturn

Individual return status for call.

Attention
ElementReturn is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
Structure	0 or 1	See “ElementReturn schema” on page 461.

ControlRecipe Type schema

A *ControlRecipe* is the same format as a master recipe, with additional information about execution.

ID

A string containing an ID of an element. In Experion, the Activity Entity prepended Tag name.

Type	# of elements expected	Example/Notes
String	1 only	*Server1:RCM_MakeResin*

BatchID

A string containing an ID of a batch. In Experion, the Client provided identification for the Activity.

Attention
Mandatory for activities of type *Batch*.

Optional for other Activity types.

Type	# of elements expected	Example/Notes
String	0 to unbounded	*Lot23*

Cluster

Server base name where Activity Entity defined.

Attention
Cluster is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	*Server1*

Asset

Activity Entity asset assignment.

Attention
Asset is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	*XYZ*

ActivityType

Activity Entity Type (*Batch* or *Procedure*).

Attention
ActivityType is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
Enumeration	0 or 1	*Batch*

PublicName

Public name associated with the Activity entity.

Attention
PublicName is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	*ResinRecipe*

ElementReturn

Individual return status for call.

Attention
ElementReturn is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
Structure	0 or 1	See “ElementReturn schema” on page 461.

BatchList schema

BatchListEntry

A string containing an ID of an element. In Experion, used to represent a single Activity.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	See “BatchListEntry schema” on page 456.

BatchListEntry schema

ID

A string containing an ID of an element. In Experion, the Activity Entity prepended Tag name.

Type	# of elements expected	Example/Notes
String	1 only	*Server1:$Activity_0930abef092-0a0a2d*

Description

A string containing a description of an element. In Experion, the description of Activity Entity.

Type	# of elements expected	Example/Notes
String	0 to unbounded	*Super strong resin*

BatchListEntryType

Identifies the type of a batch list entry. Supported enumerations are:

- *Batch* (Used for batch-type activities.)
- *Other* (Used for procedure-type activities. The *OtherValue* attribute will be set to *Procedure*.)

Type	# of elements expected	Example/Notes
Enumeration	1 only	*Batch*

Status

Identifies the status of an element. In Experion, this is represented by the sub-state of an activity.

Type	# of elements expected	Example/Notes
String	0 or 1	Running

RecipeID

A string containing an ID of a recipe. In Experion, the Activity entity prepended tag name supplied to create the Activity from.

Type	# of elements expected	Example/Notes
String	0 or 1	*Server1:RCM_MakeResin*

BatchID

A string containing an ID of a batch.

Type	# of elements expected	Example/Notes
String	0 or 1	*Lot23*

ActualStartTime

A date and time defining an actual start time of a batch or batch list entry. In Experion, the Activity start date and time.

Type	# of elements expected	Example/Notes
XML Date/Time	0 or 1	*2011-12-15T13:24:00.000+11:00* or *null*

ActualEndTime

A date and time defining an actual end time of a batch or batch list entry. In Experion, the Activity end date and time.

Type	# of elements expected	Example/Notes
XML Date/Time	0 or 1	*2011-12-16T03:11:00.000+11:00* or *null*

Cluster

Server base name where Activity Entity defined.

Attention
Cluster is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	*Server1*

Asset

Activity Entity asset assignment.

Attention
Asset is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	*XYZ*

PublicName

Public name associated with the Activity entity.

Attention
PublicName is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
String	0 or 1	*ResinRecipe*

ElementReturn

Individual return status for call.

Attention
ElementReturn is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
Structure	0 or 1	See “ElementReturn schema” on page 461.

Formula Element schema

Parameter

Defines a parameter value used in a recipe, equipment, or batch list entry. Each parameter has an ID, a type, and indication of scale; an application defined subtype, a scale reference, and may have a value. In Experion, each *Parameter* structure contains an individual Activity entity metadata parameter.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	See “Parameter Element schema” on page 458.

Parameter Element schema

ID

A string containing an ID of an element. In Experion, the process value ID.

Type	# of elements expected	Example/Notes
String	1 only	*Data.Temp.value*

Description

A string containing a description of an element. In Experion, the friendly name for the parameter as shown in the Activity data User Interface.

Type	# of elements expected	Example/Notes
String	0 or 1	Boiler Temperature

ParameterType

Identifies the type of a parameter.

This may be either a standard type or an application-specific extended type. Standard enumerations are:

- ProcessInput
- ProcessOutput
- ProcessParameter[2]

[2] Not currently supported in Experion.

- Other

If *Other*, then the type is an application-specific extension and the value is defined in the attribute *OtherValue*.

For Release 431 of Experion, the parameter types are *ProcessInput*, *ProcessOutput*, or *Header*.

Attention
Use *Other* for the *Header* parameter and set the *OtherValue* attribute to *Header*.

Type	# of elements expected	Example/Notes
Enumeration	1 only	ProcessInput

ParameterSubType

A string used to enhance sorting and filtering of parameters. In Experion, used to indicate if the parameter is editable. Returns the string of *Read Only* or *Editable*.

Type	# of elements expected	Example/Notes
String	0 to unbounded	Read Only

Value

Defines a value, including how the value should be interpreted, the data type of the value, and the unit of measure of the value. In Experion, used as a structure to hold a value.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	See “Value Element schema” on page 459.

Parameter

Defines a parameter value used in a recipe, equipment, or batch list entry. Each parameter has an ID, a type, and indication of scale; an application defined subtype, a scale reference, and may have a value. In Experion, used to specify the default, maximum, and minimum parameters associated with the parameter specifying the process value.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	Recursive structure. ***See this table definition.***

Value Element schema

ValueString

A string containing the actual value of a value. In Experion, Null if a reference has been passed to the parameter.

Type	# of elements expected	Example/Notes
Structure	1 to unbounded	*Data.Weight.value*

DataInterpretation

Identifies how to interpret a data value.

This may be either a standard type or an application-specific extended type. Standard enumerations are:

Type	Treats the value as
Constant	a constant value.
Reference	a reference to data element in the procedure.
Equation	an equation to be solved before being acted on.
External	a reference to a value external to the procedure.
Other	an application-specific extension. The value is defined in the *OtherValue* attribute.

In Experion, use *Reference* when referring to another parameter on the point. Otherwise, use *Constant*.

Type	# of elements expected	Example/Notes
Enumeration	1 only	Reference

DataType

An identification of the type of a parameter. In Experion, the data type of the metadata value. Types are based on the W3C types defined in the XSD specification, with the addition of *Enumeration* for user-defined enumerations and *Other* for application-specific extended types.

If *Other*, then the type is an application-specific extension and the value is defined in the attribute *OtherValue*.

Type	# of elements expected	Example/Notes
Enumeration	1 only	String

UnitOfMeasure

A string containing a unit of measure. Standard units of measure are not defined in this standard, but ISO standards should be used when applicable. Defined in B2MML Common Types.

In Experion, units for the metadata parameter.

Type	# of elements expected	Example/Notes
String	1 only	Fahrenheit

ElementReturn

Individual return status for call.

Attention
ElementReturn is a custom extended element specific to Experion and is not part of the BatchML schema.

Type	# of elements expected	Example/Notes
Structure	0 or 1	See “ElementReturn schema” on page 461.

BatchML schema extensions

The following are extensions specific to Experion. They are not part of the BatchML schema.

Related topics

"ReturnInformation schema" on page 461
"ElementReturn schema" on page 461
"Experion PointParam schema" on page 461
"Experion Point schema" on page 462

ReturnInformation schema

ReturnCode

Contains a return code for action.

Type	# of elements expected	Example/Notes
String	0 or 1	*8574*

ReturnString

Contains a return string for action.

Type	# of elements expected	Example/Notes
String	0 or 1	*Partial Fail - check individual element returns*

ElementReturn schema

ReturnCode

Contains a return code for action.

Type	# of elements expected	Example/Notes
String	0 or 1	0

ReturnString

Contains a return string for action.

Type	# of elements expected	Example/Notes
String	0 or 1	*Success*

Experion PointParam schema

Point

Experion structure representing a single point.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	See “Experion Point schema” on page 462.

Version

Experion version for extensions made to BatchML structures.

Type	# of elements expected	Example/Notes
String	0 to 1	*1*

ReturnInformation

Experion structure containing overall call status.

Type	# of elements expected	Example/Notes
Structure	1 only	See “ReturnInformation schema” on page 461.

Experion Point schema

ID

Experion string containing an ID of an element defining a single point.

Type	# of elements expected	Example/Notes
String	0 to 1	*Server1:$Activity_0930abef092-0a0a2d*

Parameter

Experion parameter to have value read.

Type	# of elements expected	Example/Notes
Structure	0 to unbounded	See “Parameter Element schema” on page 458.

BatchML Object Model class diagrams

This section contains the BatchML class definitions and types used by Batch Application Services.

Attention
Only those data members that are used by Batch Application Services appear in these class diagrams. They do not necessarily depict the complete BatchML structure.

ActivityTypeType

ActivityTypeType
+Value : string

Figure 56: ActivityTypeType

ActualEndTimeType

ActualEndTimeType
+Value : System.DateTimeOffset

Figure 57: ActualEndTimeType

ActualStartTimeType

ActualStartTimeType
+Value : System.DateTimeOffset

Figure 58: ActualStartTimeType

BatchIDType

BatchIDType
+Value : string

Figure 59: BatchIDType

BatchInformationType

BatchInformationType
+BatchList : BatchListType +ControlRecipe : ControlRecipeType +Description : DescriptionType +EnumerationSet : EnumerationSetType +MasterRecipe : MasterRecipeType +ReturnInformation : ReturnInformationType +Version : string

Figure 60: BatchInformationType

BatchListEntryType

BatchListEntryType
+ActualEndTime : ActualEndTimeType +ActualStartTime : ActualStartTimeType +Asset : string +BatchID : BatchIDType +BatchListEntryType1 : BatchListEntryTypeType +Cluster : string +CreateTime : DateTimeType +Description : DescriptionType +ElementReturn : ReturnInformationType +ExecutionStatus : string +ID : IDType +Mode : ModeType +ModeAttribute : string +PublicName : string +RecipeID : RecipeIDType +Stage : string +Status : BatchStatusType

Figure 61: BatchListEntryType

BatchListEntryTypeType

BatchListEntryTypeType
+OtherValue : string +Value : string

Figure 62: BatchListEntryTypeType

BatchListType

BatchListType
+BatchListEntry : BatchListEntryType

Figure 63: BatchListType

BatchParameterType

BatchParameterType
+Description : DescriptionType +ElementReturn : ReturnInformationType +ID : IDType +IsPrimaryUnit : bool +IsPrimaryUnitSpecified : bool +Parameter : BatchParameterType +ParameterSubType : ParameterSubTypeType +ParameterType : ParameterTypeType +Value : BatchValueType

Figure 64: BatchParameterType

BatchStatusType

BatchStatusType
+OtherValue : string +Value : string

Figure 65: BatchStatusType

BatchValueType

BatchValueType
+DataIntrepretation : DataIntrepretationType +DataType : DataTypeType +Description : string +EnumerationSetID : EnumerationSetIDType +UnitOfMeasure : UnitOfMeasureType +ValueString : ValueStringType

Figure 66: BatchValueType

ControlRecipeType

ControlRecipeType
+BatchID : BatchIDType +ID : IDType

Figure 67: ControlRecipeType

DataIntrepretationType

DataIntrepretationType
+OtherValue : string +Value : string

Figure 68: DataIntrepretationType

DataTypeType

DataTypeType
+OtherValue : string +Value : string

Figure 69: DataTypeType

DateTimeType

`DateTimeType` has two class definitions: one for C# and one for C++.

DateTimeType
-Value : System.DateTimeOffset

Figure 70: DateTimeType - C#

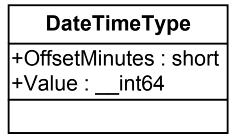


Figure 71: DateTimeType - C++

DescriptionType

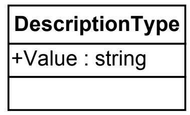


Figure 72: DescriptionType

EnumerationSetIDType

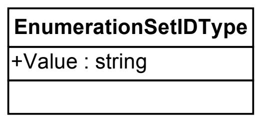


Figure 73: EnumerationSetIDType

EnumerationSetType

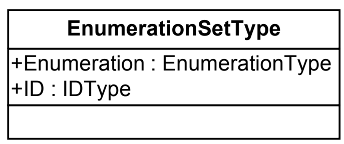


Figure 74: EnumerationSetType

EnumerationStringType

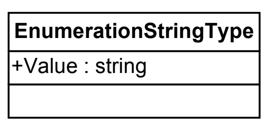


Figure 75: EnumerationStringType

EnumerationType

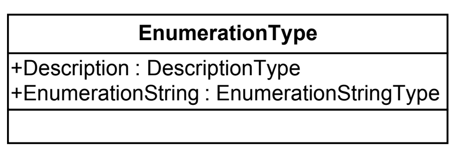


Figure 76: EnumerationType

FormulaType

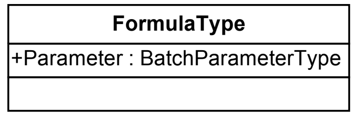


Figure 77: FormulaType

IDType

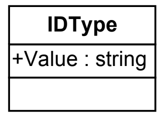


Figure 78: IDType

MasterRecipeType

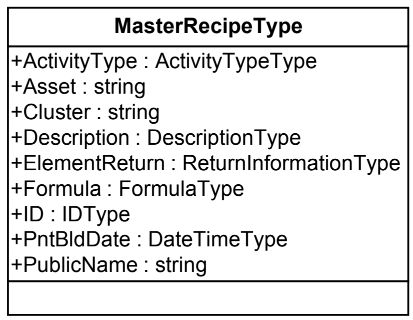


Figure 79: MasterRecipeType

ModeType

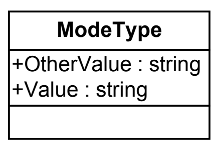


Figure 80: ModeType

ParameterSubTypeType

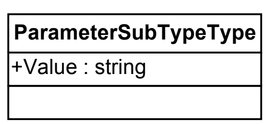


Figure 81: ParameterSubTypeType

ParameterTypeType

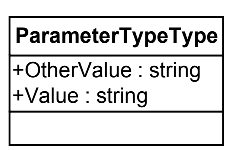


Figure 82: ParameterTypeType

PointParamType

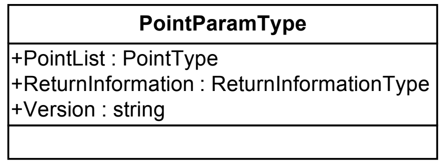


Figure 83: PointParamType

PointType

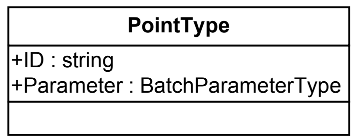


Figure 84: PointType

RecipeIDType

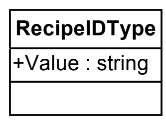


Figure 85: RecipeIDType

ReturnInformationType

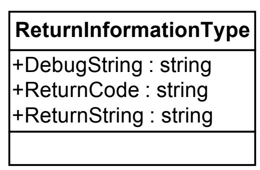


Figure 86: ReturnInformationType

StatusType

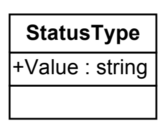


Figure 87: StatusType

UnitOfMeasureType

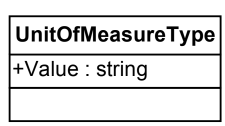


Figure 88: UnitOfMeasureType

ValueStringType

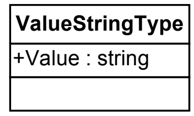


Figure 89: ValueStringType

VersionType

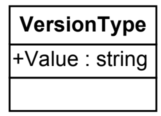


Figure 90: VersionType

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.

Related topics

"Server Automation Object Model" on page 472
"HMIWeb Object Model" on page 473
"Station Scripting Objects" on page 474
"Station Object Model" on page 477

Server Automation Object Model

The Server Automation Object Model represents the server, points, events, assets and reports.

Notes

- In Visual Basic, choose **Project** > **References**, 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 477.

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 page 473.

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*.

Related topics

“Implementing a Detach method” on page 476

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 **File** > **Make 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:

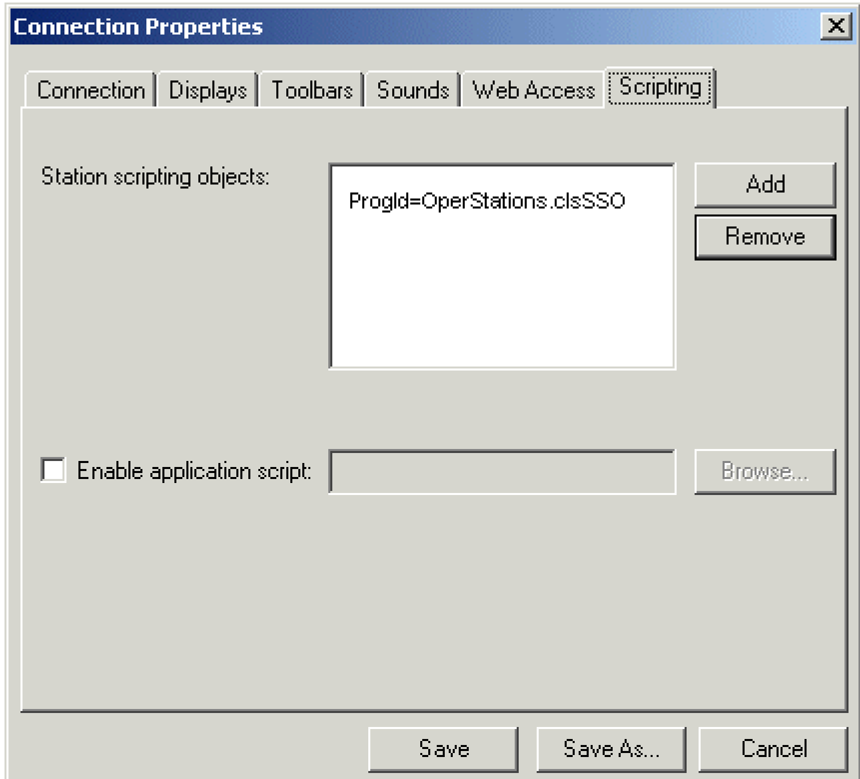
   a. Copy the SSO to the Station computer and register it. See “Registering an SSO” on page 475. (You can skip this step if you compiled the SSO on the computer because it automatically registers itself if the compilation is successful.)

   b. Choose **Station** > **Connection 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 multiple window Station support

If you wish to write a SSO that utilizes the multi-window Station interfaces then you must set the objectSupportsMultiWindow property to true and expose it so that the SSO will work in a multiple window Station.

Attention

The key difference is that single window SSOs will be passed IApplication and multi-window SSOs will be given IStation. IStation has extra methods that allow developers to write code that handles Station having more than one display open.

Visual Basic example:

```
' Set this property to True if SSO need to work in both
' single window and multi-window Station
Public Property Get objectSupportsMultiWindow() As Boolean
  objectSupportsMultiWindow = True
End Property
```

Implementing SetMultiWindowStationObject

The SetMultiWindowStationObject method must be implemented so that Station will create your SSO. As soon as Station has created your SSO, it calls SetMultiWindowStationObject, passing in a reference to Station's Station object.

You may use this method to initialize a global variable in your SSO that it references to Station's Station object.

Visual Basic example:

```
Dim withEvents m_objStation As Station.Station
Public Sub SetMultiWindowStationObject(ByRef objStation
As Station.Station)
  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
```

Related topics

“Station Scripting Objects” on page 474

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 473.
- In order to control DSP displays, you must first create Station using the Application object of the “HMIWeb Object Model” on page 473 (see “HMIWeb Object Model” on page 473). 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 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

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.

BatchML An XML representation of the S88 standard.

CBR Class Based Recipe.

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 as controller. Channels are defined using the Quick Builder tool.

CIM Communications Interface Module

CNI card ControlNet EISA Interface card.

collection A collection is a set of named values or *display objects* that are used in *scripts*.

COM Component Object Model

Control Builder The control building software for the Process Controller.

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 Controllers The term used to refer to all control hardware (chassis, power supply, Control Processor, and ControlNet bridge) as a single entity in an Experion system.

Process software An umbrella term for Control Builder and other hybrid controller software.

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.

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®, SafeBrowse®, TotalPlant®, and TDC 3000® are registered trademarks of Honeywell International, Inc.

OneWireless™ is a trademark 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

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 Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

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.

  or

- Contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” 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 on Experion PKS. These classes are taught by experts in the field of process control systems. For more information about these classes, contact your Honeywell representative, or see http://www.automationcollege.com.

Index

A

action algorithms
Status Change Task Request algorithm, Action Algo 69 41
activating
tasks
when a display is called up 42
while a point is on scan 40
activities
configuring an activity before starting it 405
creating an activity 399
filters
actual end time 421
actual start time 420
asset 414
batch ID 415
create time 419
public name 416
server base name 414
stage 417
state 418
tag name 417
type 413
monitoring the state of the activity 409
retrieving metadata information about an activity 400
returning only one activity 400
running an activity through its life cycle 407
activity entities
filters
asset 429
point build date 431
public name 431
server base name 430
type 429
getting a list of 396
getting more information about 397
ActivityAssetFilter 369
ActivityEntityFilterList 369
ActivityFilterList 369
ActivityTypeFilter 369
ActivityTypeType 463
actual end time filter 421
actual start time filter 420
ActualEndTimeType 463
ActualStartTimeType 463
adding
ADDTSK 84
application task 84
dependencies 385
directories 385
ADDTSK 84
alarms
generating from task 69
algorithms
Status Change Task Request algorithm, Action Algo 69 41
APIs
Batch Application Services 347
Network
about 274
error codes 283
function summary 276
functions 291
parameter numbers, determining 278
point numbers, determining 277
Server
development environment, about 15
error codes 22
functions, alarm or event or message, notification s tructure 142, 144, 149
functions, alarm or event or message, send 156
functions, alarm or event or message, structure and send by type 99
functions, database, file, lock 154
functions, database, file, unlock 207
functions, database, record, lock 155
functions, database, record, unlock 208
functions, error code, retrieval 120
functions, IEEE 754 special value, Infinity creatio n 211, 212
functions, IEEE 754 special value, Infinity validati on 220
functions, IEEE 754 special value, NaN creation 226, 227
functions, IEEE 754 special value, NaN validation 221
IEEE 754 validation functions 23
reference 13
app.config file 381
applications
C/C++ application template 27
template, in C/C++ 27
types of applications you can develop 26
argc parameter 28
argv parameter 28
asset filter 414, 429
attribute names and their index values 144
automated system account 351
Automation Objects, overview 471

B

backward-compatible functions 256

Batch Application Services 347
- automated system account 351
- BatchML Object Model 352
- Client Component, configuring 443
- Client Component, installing 440
- connections 373
- Console Stations 351
- copying resources 391
- CreateActivity 361
- development environment 349
- error codes 373
- event logging 351
- filtering 369
- filtering class diagrams 369
- function reference 353
- GetActivityEntityList 356
- GetActivityEntityMetadata 358
- GetActivityList 353
- GetPointParameterValues 365
- header file 393
- IsPrimary 353
- licensing 350
- logging events 351
- namespaces 393
- overview 348
- reference 451
- resources 391
- SetPointParameterValues 363
- topology 348
- tutorial 375

Batch Application Services Client Component
- installing 440

batch ID filter 415
BatchIDType 463
BatchInformation schema 452
BatchInformationType 463
BatchList schema 456
BatchListEntry schema 456
BatchListEntryType 463
BatchListEntryTypeType 463
BatchListType 463
BatchML Object Model 352
- class diagrams 463
- schema 452, 461

BatchParameterType 463
BatchStatusType 463
BatchValueType 463

C

C language
- include files for server global definitions 20, 288
- include files, placement of 27

C routine
- assigns LRN to current thread, AssignLrn() 101
- backward-compatible functions 256
- c_almmsg_...() 99
- control a list of point parameter values, hsc_param_values_put() 184
- control a point parameter value, hsc_param_value_put() 182
- convert a C string to a FORTRAN string, ctofstr() 103
- convert a character string to upper case, c_upper() 252
- convert a FORTRAN string to a C string, ftocstr() 115
- convert a GUID from binary to string, hsc_StringFromGUID() 206
- convert between Julian days and Gregorian date, JULIAN 223
- convert date/time from HSCTIME format to FILETIME format, HscTimeToFiletime() 210
- convert date/time from HSCTIME format to VARIANT DATE format, HscTimeToDate() 209
- converts a GUID from string to binary, hsc_GUIDFromString() 141
- copy an integer array to a character string, c_intchr() 215
- copy character buffer to integer buffer, c_chrint 102
- database file access routines, c_dataio_...() 104
- destroy sn enumlist, hsc_enumlist_destroy() 140
- determine whether a returned GDA status value is a warning, IsGDAwarning() 218
- determine whether a returned GDA status value is an error, IsGDAerror() 216
- determine whether a returned GDA status value is neither an error nor a warning, IsGDAnoerror() 217
- determines if a returned status value is an error, hsc_IsError() 152
- determines if returned status value is a warning, hsc_IsWarning() 153
- examples 269
- execute command line, c_ex() 114
- find the Station number of a display task given the task's LRN, stn_num() 244
- finds LRN of display task for Station, dsply_lrn() 113
- free the memory used to hold a list of points, hsc_em_FreePointList() 135
- get a list of parameters, hsc_param_list_create() 168
- get a parameter name, hsc_param_name() 170
- get a parameter number, hsc_param_number() 171
- get a parameter's data type, hsc_param_type() 174
- get a parameter's format, hsc_param_format() 163
- get a point parameter value of specified type, hsc_param_value_of_type() 178
- get a point parameter value, hsc_param_value() 176
- get a point type name, hsc_pnttyp_name() 190
- get a point type number, hsc_pnttyp_number() 191
- get a point's name, hsc_point_name() 203
- get a point's number, hsc_point_number() 204
- get a point's type, hsc_point_type() 205
- get an enumerated list of parameter values, hsc_param_enum_list_create() 159
- get an enumeration string, hsc_param_enum_string() 162
- get an enumeration's ordinal value, hsc_param_enum_ordinal() 161
- get application record for task, c_getapp() 118
- get database control request, c_gdbcnt() 117
- get history interface parameters, c_gethstpar_..._2() 121
- get logical resource number, c_getlrn() 124

get multiple point parameter values, hsc_param_values( ) 179
get parameter data entry limits, hsc_param_limits() 165
get parameter data range, hsc_param_range() 172
get parameters from queued task request, c_getprm() 126
get parameters from task request block, c_getreq() 128
get the asset ancestors for an asset, hsc_asset_get_ancestors() 131
get the children of an asset, hsc_asset_get_children() 132
get the descendents of an asset, hsc_asset_get_descendents() 133
get the history interface parameters, c_gethstpar() 258
get the last time a point was changed, hsc_em_GetLastPointChangeTime() 136
get the parents of an asset, hsc_asset_get_parents() 134
get the point numbers for all root entities, hsc_em_GetRootEntities() 139
get the point numbers for root assets, hsc_em_GetRoot_Assets() 138
get the point numbers of the root Alarm Groups, hsc_em_GetRootAlarmGroups() 137
get values of a list of points, c_getlst() 125
give values to a list of points, c_givlst() 130
global common load, c_gbload() 116
insert a character string into a PARvalue union, StrtoPV() 245
insert a double value into a PARvalue union, DbletoPV( ) 112
insert a priority and sub-priority value into a PARvalue union, PritoPV() 234
insert a real value into a PARvalue union, RealtoPV() 236
insert a time value into a PARvalue union, TimetoPV() 246
insert an int2 value into a PARvalue union, Int2toPV() 213
insert an int4 value into a PARvalue union, Int4toPV() 214
list all point types, hsc_pnttyp_list_create() 188
lock a database file, hsc_lock_file() 154
lock a record of a database file, hsc_lock_record() 155
mark a task for deletion, c_deltsk() 111
process a point special, c_pps_2() 230
process a point special, c_pps() 261
process a point special, c_ppsw_2() 231
process a point special, c_ppsw() 262
process a point value, c_ppv_2() 232
process a point value, c_ppv() 263
process a point value, c_ppvw_2() 233
process a point value, c_ppvw() 264
pulse the watchdog timer for a task, c_wdon() 253
queue file to print system, c_prsend_...() 235
removes current LRN assignment for thread, DeassignLrn() 110
request a task is not busy, c_rqtskb...() 237
return a list of points that refer to the specified point, hsc_point_get_referers() 201
return a list of points to which a specified point refers, hsc_point_get_references() 200
return all children, hsc_point_get_children() 194
return all containment ancestors above a specified point, hsc_point_get_containment_ancestors() 195
return all containment children for a specified point, hsc_point_get_containment_children() 196
return all containment descendents above a specified point, hsc_point_get_containment_parents() 198
return all containment descendents below a specified point, hsc_point_get_containment_descendents() 197
return all parents, hsc_point_get_parents() 199
return error code from DGAERR, GetGDAERRcode() 119
return the entity name of a point, hsc_point_entityname( ) 192
return the full item name of the point, hsc_point_fullname() 193
return the GUID for the specified point, hsc_point_guid( ) 202
return the INFINITY value as a Double data type, infdouble() 211
return the INFINITY value as a Float data type, inffloat( ) 212
return the QNaN value as a Double data type, nandouble() 226
return the QNaN value as a Float data type, nanfloat() 227
save a point parameter value, hsc_param_value_save() 186
scan a point special, c_sps_2() 239
scan a point special, c_sps() 265
scan a point special, c_spsw_2() 240
scan a point special, c_spsw() 266
scan a point value, c_spv_2() 241
scan a point value, c_spv() 267
scan a point value, c_spvw_2() 242
scan a point value, c_spvw() 268
send a message to Station, c_oprstr_...() 228
send a notification structure to raise or normalize and alarm, event, or message, hsc_notif_send() 156
set an attribute value of a notification structure, hsc_insert_attrib_byindex() 149
set an attribute value of a notification structure, hsc_insert_attrib() 142, 144
start a timer for the calling task, c_tmstart() 248
start the watchdog timer for the calling task, c_wdstrt() 254
stop a timer for the calling task, c_tmstop() 247
subscribe to a list of point parameters, hsc_param_subscribe() 167
terminate task with error status and modify restart address, c_trm04() 249
terminate task with error status, c_trmtsk() 250
test a real value for -0.0, ic_mzero() 225
test a task's status, c_tstskb() 251
test for a bad parameter value, c_badpar() 257
unlock a database file, hsc_unlock_file() 207
unlock a record of a database file, hsc_unlock_record() 208

update Controller's sample time counter, c_stcupd() 243
validate the Double data type argument as a NaN value, isnandouble() 221
validate the Double data type argument as an Infinity va lue, isinfdouble() 219
validate the Float data type argument as a NaN value, is nanfloat() 222
validate the Float data type argument as an Infinity valu e, isinffloat() 220
wait for a task to become dormant, c_wttskb() 255
write a message to the log file, c_logmsg() 224
c_almmsg_...(), Server API function 99
c_geterrno(), Server API function 120
CBR
see class based recipes 447
circular files, accessing with functions 50
class diagrams
Batch Application Services
filtering 369
BatchML Object Model 463
classed based recipes 447
client proxies
communicating with redundant servers 445
combining multiple criteria in string filters 422
commands and utilities
described 26
development utilities 83
types of utilities you can develop 26
configuring
an activity before starting it 405
Batch Application Services Client Component 443
client application to redirect messages to the redundancy service 444
connections
Batch Application Services 373
to multiple servers 438
Console Stations
Batch Application Services API calls 351
consuming the WCF Service 377
controllers
unsupported devices 73
ControlRecipe Type schema 454
copying
Batch Applications Services resources 391
countdown timer
function of 45
resetting 45
create time filter 419
CreateActivity 361
creating
a new C# project 376
a new C++ project 383
an activity 399
C# project 376
C++ project 383
new project 376, 383
CriteriaAnd 369
CriteriaAsset 369
CriteriaDateTime 369
CriteriaOr 369
CriteriaString 369
CT (create task) 85

D

data types
C application 29
defined in header file defs.h 29
databases
about 48
data folder contents 49
described 48
file structure
logical 50
introduction 48
object-based real-time database files 53
physical structure 49
scanning the database 66
server database 49
server software 66
DataIntrepretationType 463
DataTypeType 463
DateTimeFilter 369
nesting criteria 424
DateTimeType 463
DBG (run a specified LRN) 87
DBSCN (database scanning) 66
deletion
marking a task for from a command line 37
marking the application for 28
dependencies
adding 385
DescriptionType 463
development environment
Batch Application Services 349
utilities 83
diagnostics
Network API functions 344
directories
adding 385
DIRTRY, definition of 60
DT (delete task) 88

E

EActivityTypes 369
EActTypeFilterModes 369
EDateTimeFilterModes 369
ElementReturn schema 461
EnumerationSetType 463
EnumerationStringType 463
EnumerationType 463
environment
Batch Application Services 349
error codes
Batch Application Services 373
Network API 283
Server API 22
error scenarios 435
EStringFilterModes 369
ETR (enter task request) 89

events
  logging
    Batch Application Services 351
exception handling 435
Experion Point schema 462
Experion PointParam schema 461

F

faulted proxy 435
FILDMP (dump/restore a logical file) 90
FILEIO (modify contents of a logical file) 91
files
  databases
    structure 49
  logical, accessing 59
  memory resident, accessing 59
  number reserved in database for application storage 61
  protecting data in shared 53
  relative type 50
  structure 50
filtering items
  activity lists and activity entity lists 412
  Batch Application Services 369
filters
  actual end time 421
  actual start time 420
  asset 414, 429
  batch ID 415
  create time 419
  point build date 431
  public name 416, 431
  server base name 414, 430
  stage 417
  state 418
  tag name 417
  type 413, 429
folders
  data 20, 288
  run 20, 288
  user source files 20, 288
Formula Element schema 458
FormulaType 463
FORTRAN
  include files for server global definitions 20, 288
function keys
  activating tasks 41
function reference
  backward compatibility 323
  Batch Application Services 353
  CreateActivity 361
  diagnostics 344
  GetActivityEntityList 356
  GetActivityEntityMetadata 358
  GetActivityList 353
  GetPointParameterValues 365
  hsc_bad_value 292
  hsc_napierrstr 293, 323
  IsPrimary 353
  rgetdat 293
  rgethstpar_date 323
  rgethstpar_ofst 323
  rgetpnt 325
  rgetpntval 328
  rgetpntval_ascii 328
  rgetval_ascii 326
  rgetval_hist 326
  rgetval_numb 326
  rhsc_notifications 295
  rhsc_param_hist_date_bynames 297
  rhsc_param_hist_dates 329
  rhsc_param_hist_dates_2 301
  rhsc_param_hist_offset_bynames 297
  rhsc_param_hist_offsets 329
  rhsc_param_numbers 338
  rhsc_param_numbers_2 304
  rhsc_param_value_bynames 306
  rhsc_param_value_put_bynames 309
  rhsc_param_value_put_sec_bynames 311
  rhsc_param_value_puts 331
  rhsc_param_value_puts_2 313
  rhsc_param_values 334
  rhsc_param_values_2 316
  rhsc_point_numbers 339
  rhsc_point_numbers_2 319
  rputdat 321
  rputpntval 341
  rputpntval_ascii 341
  rputval_ascii 342
  rputval_hist 341
  rputval_numb 342
  SetPointParameterValues 363
functions
  accessing parameter values by name 283
  backward-compatible functions 256
  IEEE 754 special values 23
  Network API 276, 291
    Server API
      c_almmsg_...() 99
      c_geterrno() 120
      hsc_insert_attrib_byindex() 149
      hsc_insert_attrib() 142, 144
      hsc_lock_file() 154
      hsc_lock_record 155
      hsc_notif_send() 156
      hsc_unlock_file() 207
      hsc_unlock_record() 208
      infdouble 211
      inffloat 212
      isinffloat 220
      isnandouble 221
      nandouble 226
      nanfloat 227

G

GetActivityEntityList 356
GetActivityEntityMetadata 358
GetActivityList 353
GetPointParameterValues 365
GETREQ (failure to call in task) 39

getting a list of activity entities 396
getting more information
about an activity entity 397

H

history
accessing block of history 58
blocks, accessing 58
files, structure in database 50
storage intervals 58
HMIWeb Object Model 473
hsc C routines
AssignLrn() 101
c_almmsg_...() 99
c_badpar() 257
c_chrint() 102
c_dataio_,,,() 104
c_deltsk() 111
c_ex() 114
c_gbload() 116
c_gdbcnt() 117
c_getapp() 118
c_geterrno() 120
c_gethstpar_..._2() 121
c_gethstpar() 258
c_getlrn() 124
c_getlst() 125
c_getprm() 126
c_getreq() 128
c_givlst() 130
c_intchr() 215
c_logmsg() 224
c_mzero() 225
c_oprstr...() 228
c_pps_2() 230
c_pps() 261
c_ppsw_2() 231
c_ppsw() 262
c_ppv_2() 232
c_ppv() 263
c_ppvw_2() 233
c_ppvw() 264
c_prsend_...() 235
c_rqtskb...() 237
c_sps_2() 239
c_sps() 265
c_spsw_2() 240
c_spsw() 266
c_spv_2() 241
c_spv() 267
c_spvw_2() 242
c_spvw() 268
c_stcupd() 243
c_tmstart() 248
c_tmstop() 247
c_trm04() 249
c_trmtsk() 250
c_tstskb() 251
c_upper() 252
c_wdon() 253
c_wdstrt() 254
c_wttskb() 255
ctofstr() 103
DbletoPV() 112
DeassignLrn() 110
dsply_lrn() 113
ftocstr() 115
GetGDAERRcode() 119
hsc_asset_get_ancestors() 131
hsc_asset_get_children() 132
hsc_asset_get_descendents() 133
hsc_asset_get_parents() 134
hsc_em_FreePointList() 135
hsc_em_GetLastPointChangeTime() 136
hsc_em_GetRootAlarmGroups() 137
hsc_em_GetRootAssets() 138
hsc_em_GetRootEntities() 139
hsc_enumlist_destroy() 140
hsc_GUIDFromString() 141
hsc_insert_attrib_byindex() 149
hsc_insert_attrib() 142, 144
hsc_IsError() 152
hsc_IsWarning() 153
hsc_lock_file() 154
hsc_lock_record() 155
hsc_notif_send() 156
hsc_param_enum_list_create() 159
hsc_param_enum_list_ordinal() 161
hsc_param_enum_string() 162
hsc_param_format() 163
hsc_param_limits() 165
hsc_param_list_create() 168
hsc_param_name() 170
hsc_param_number() 171
hsc_param_range() 172
hsc_param_subscribe() 167
hsc_param_type() 174
hsc_param_value_of_type() 178
hsc_param_value_put() 182
hsc_param_value_save() 186
hsc_param_value() 176
hsc_param_values_put() 184
hsc_param_values() 179
hsc_pnttyp_list_create() 188
hsc_pnttyp_name() 190
hsc_pnttyp_number() 191
hsc_point_entityname() 192
hsc_point_fullname() 193
hsc_point_get_children() 194
hsc_point_get_containment_ancestors() 195
hsc_point_get_containment_children() 196
hsc_point_get_containment_descendents() 197
hsc_point_get_containment_parents() 198
hsc_point_get_parents() 199
hsc_point_get_references() 200
hsc_point_get_referers() 201
hsc_point_guid() 202
hsc_point_name() 203
hsc_point_number() 204

hsc_point_type() 205
hsc_StringFromGUID() 206
hsc_unlock_file() 207
hsc_unlock_record() 208
hscTimeToDate() 209
hscTimeToFiletime() 210
infdouble() 211
inffloat() 212
int2toPV() 213
int4toPV() 214
IsGDAerror() 216
IsGDAnoerror() 217
IsGDAwarning() 218
isinandouble() 221
isinanfloat() 222
isinfdouble() 219
isinffloat() 220
nandouble() 226
nanfloat() 227
PritoPV() 234
RealtoPV() 236
stn_num() 244
StrtoPV() 245
TimetoPV() 246
hsc_bad_value 292
hsc_napierrstr 293, 323

I

identifying
parameters 55
points 55
IDType 463
IEEE 754 special values 23
INF validation functions 23
infdouble, Server API function 211
inffloat, Server API function 212
installing
Batch Application Services Client Component 440
intermediate tutorial 412
activity entity filter
asset 429
point build date 431
public name 431
server base name 430
type 429
activity filter
actual end time 421
actual start time 420
asset 414
batch ID 415
create time 419
public name 416
server base name 414
stage 417
state 418
tag name 417
type 413
combining multiple criteria in string filters 422
filtering activity lists and activity entity lists 412
nesting criteria for DateTimeFilter 424
printing returned activities 426
printing returned activity entities 432
isinffloat, Server API function 220
isnandouble, Server API function 221
IsPrimary 353
IsPrimary operation 394

L

large list operations, supporting 381
licensing
Batch Application Services 350
lists, point 55
log files
viewing 28
logging events
Batch Application Services 351
logical file structure 50
logical files, accessing 59
LRNs
for a task 36
identifying with GETLRN 36

M

managing
redundancy 439
MasterRecipe Type schema 453
MasterRecipeType 463
memory resident files, accessing 59
messages
Message Zone, location of in Station 70
types available for display 70
metadata
retrieving from an activity 400
ModeType 463
Monitoring activities 409
multiple servers, connecting to 438
multithreading, Server API, tasks 21

N

NaN validation functions 23
nandouble, Server API function 226
nanfloat, Server API function 227
nesting criteria for DateTimeFilter 424
Network API
about 274
backward-compatibility functions 323
diagnostics 344
error codes 283
function reference
rhsc_param_hist_offsets_2 302
functions 276, 291
hsc_bad_value 292
hsc_napierrstr 293, 323
parameter numbers, determining 278
point numbers, determining 277
rgetdat 293

rgethstpar_date 323
rgethstpar_ofst 323
rgetpnt 325
rgetpntval 328
rgetpntval_ascii 328
rgetval_ascii 326
rgetval_hist 326
rgetval_numb 326
rhsc_notifications 295
rhsc_param_hist_date_bynames 297
rhsc_param_hist_dates 329
rhsc_param_hist_dates_2 301
rhsc_param_hist_offset_bynames 297
rhsc_param_hist_offsets 329
rhsc_param_hist_offsets_2 302
rhsc_param_numbers 338
rhsc_param_numbers_2 304
rhsc_param_value_bynames 306
rhsc_param_value_put_bynames 309
rhsc_param_value_put_sec_bynames 311
rhsc_param_value_puts 331
rhsc_param_value_puts_2 313
rhsc_param_values 334
rhsc_param_values_2 316
rhsc_point_numbers 339
rhsc_point_numbers_2 319
rputdat 321
rputpntval 341
rputpntval_ascii 341
rputval_ascii 342
rputval_hist 341
rputval_numb 342
Network Server Option 274

O

Object Models
HMIWeb 473
overview 471
Server Automation 472
Station (DSP displays) 477
Station Scripting Objects (SSOs) 474
object-based real-time database files 53
ODBC client, developing 33
OPC Client
developing 32
operating system file, queuing for printing 71
options
Network Server Option 274

P

parameter block
calling with GETREQ 28
Parameter Element schema 458
parameters
accessing values 55, 283
argc parameter 28
argv parameter 28
functions for accessing parameter values by name 283
identifying 55
number determining using Network API 278
using argv and argc in C application 28
ParameterSubTypeType 463
ParameterTypeType 463
partial function fail, handling 435
periodic requests, stopping 39
point build date filter 431
point history, accessing 58
point lists 55
point parameter
preventing periodic scanning of 56
retrieving values 400
point parameters
retrieving large amounts from user tables with DBSCN 66
PointParamType 463
points
identifying 55
number determining using Network API 277
PointType 463
portability, achieving with macros in C 29
printf, avoiding in C files 30
printing
returned activities 426
returned activity entities 432
to a Station printer 71
projects
creating a new project 376, 383
proxies
client proxies for communicating with redundant servers 445
public name filter 416, 431
PV
algorithms
Cyclic Task Request algorithm, PV Algo 16 40

Q

queue, flushing a request 39

R

RecipeIDType 463
recipes
class based recipes 447
redundancy
configuring client application to redirect messages to the redundancy service 444
installing the Batch Application Services Client Compo nent 440
managing 439
using client proxies to communicate with redundant serv ers 445
redundant servers 31
REMTSK (remove application task) 92
reports
activating a task 42
configuring to request a task 42
requests, flushing from queue with GETREQ 39

resources
copying Batch Application Services resources 391
ReturnInformation schema 461
ReturnInformationType 463
returning only one activity 400
rgetdat 293
rgethstpar_date 323
rgethstpar_ofst 323
rgetpnt 325
rgetpntval 328
rgetpntval_ascii 328
rgetval_ascii 326
rgetval_hist 326
rgetval_numb 326
rhsc_notifications 295
rhsc_param_hist_date_bynames 297
rhsc_param_hist_dates 329
rhsc_param_hist_dates_2 301
rhsc_param_hist_offset_bynames 297
rhsc_param_hist_offsets 329
rhsc_param_hist_offsets_2 302
rhsc_param_numbers 338
rhsc_param_numbers_2 304
rhsc_param_value_bynames 306
rhsc_param_value_put_bynames 309
rhsc_param_value_put_sec_bynames 311
rhsc_param_value_puts 331
rhsc_param_value_puts_2 313
rhsc_param_values 334
rhsc_param_values_2 316
rhsc_point_numbers 339
rhsc_point_numbers_2 319
rputdat 321
rputpntval 341
rputpntval_ascii 341
rputval_ascii 342
rputval_hist 341
rputval_numb 342
RQTSKB routine 44
running
batch activity through its life cycle 407
task when a display is called up 42

S

schemas
BatchInformation 452
BatchList 456
BatchListEntry 456
BatchML 452, 461
ControlRecipe Type 454
ElementReturn 461
Experion Point 462
Experion PointParam 461
Formula Element 458
MasterRecipe Type 453
Parameter Element 458
ReturnInformation 461
Value Element 459
server
connecting to multiple servers 438
redundancy 31
server base name filter 414, 430
Server API 13
development environment, about 15
error codes 22
functions
alarm or event or message, notification structure 142, 144, 149
alarm or event or message, send 156
alarm or event or message, structure and send by type 99
database, file, lock 154
database, file, unlock 207
database, record, lock 155
database, record, unlock 208
error code, retrieval 120
IEEE 754 special value, Infinity creation 211, 212
IEEE 754 special value, Infinity validation 220
IEEE 754 special value, NaN creation 226, 227
IEEE 754 special value, NaN validation 221
IEEE 754 validation functions 23
tasks
multithreading 21
Server Automation Object Model 472
server base name filter 414, 430
server databases
structure 49
servers
database 49
logical file structure 50
databases
structure 49
logical file structure 50
SetPointParameterValues 363
simple tutorial 394
configuring an activity before starting it 405
creating an activity 399
getting a list of activity entities 396
getting more information about an activity entity 397
monitoring the state of the activity and reacting to state changes 409
querying the IsPrimary operation 394
retrieving metadata information about an activity 400
retrieving values of point parameters 400
returning only one activity 400
running an activity through its life cycle 407
source code files
folder for 20, 288
SPVW, use of 56
SSOs (Station Scripting Objects 474
stage filter 417
state filter 418
Station Object Model (DSP displays) 477
Station Scripting Objects (SSOs) 474
Stations
running a task 68
tasks 68
Status Change Task Request algorithm, Action Algo 69 41
status point

activating a task 41
change of state 41
StatusType 463
STCUPD, example use 76
StringFilter 369
strings 53
converting a C string to a FORTRAN string 103
converting a FORTRAN string to a C string 115
copying to integer buffer 102
logical file structure, in 53
supporting large list operations 381
SYSBLD, edit intervals 58

T

tag name filter 417
TAGLOG (list point information) 93
tasks
activating
on a regular basis 39
while a point is on scan 40
with a pull-down menu item 42
with function keys 41
with Station display pushbutton 42
alarm generating 69
blocking with TRM04 call (in C) 28
choosing an LRN 36
described 26
including an operator prompt 42
marking for deletion from a command line 37
running from Station 68
Server API
multithreading 21
testing status of 44
types of tasks you can develop 26
writing
for unsupported controller 73
messages to log file 28
templates
C/C++ application template 27
timer table, making entries in with TMSTRT 39
topology 348
tutorials
Batch Application Services 375
C# project 376
C++ project 383
intermediate 412
prerequisites 376, 383
simple 394
type filter 413, 429

U

UnitOfMeasureType 463
user (Station)
automated system account 351
user scan tasks
database design 75
example 76
scanning 75
user tables
circular structure 61
configuring 61
description and use of 61
displaying data 61
how to set up 49
layout for Network API 280
modifying data 61
modifying with utbbld 61
preconfigured by server 61
reading entire record from 280
reading one field from 280
relative (direct) 61
types of 61
UTBBLD utility 62
writing whole record to 280
USRLRN (list available task LRNs) 36, 94
UTBBLD
system status checks 65
user table utility 61, 62
UTC offset for DateTime filtering 372
utilities
described 26
development utilities 83
types of utilities you can develop 26
USRLRN, list available tasks 36

V

valid attributes for a category 146
Value Element schema 459
ValueStringType 463
VersionType 463

W

watchdog timers, described 45
WCF
configuring client application to redirect messages to the redundancy service 444
consuming the WCF Service 377
WDSTRT (watchdog timer start routine) 45
WTTSKB routine 44