

Experion PKS  
HCI/OPC Data Access User's Guide

EPDOC-XX52-en-431A  
February 2015

**Release 431**

Document	Release	Issue	Date
EPDOC-XX52-en-431A	431	0	February 2015

## Disclaimer

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

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

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

Copyright 2015 - Honeywell International Sàrl

# Contents

<b>1 About This Document .....</b>	<b>7</b>
<b>2 Getting Started .....</b>	<b>9</b>
2.1 Introduction .....	10
2.1.1 Assumptions .....	10
2.1.2 Topics .....	10
2.1.3 System .....	10
2.1.4 References .....	11
2.1.5 About This Guide .....	11
<b>3 Functional Overview .....</b>	<b>13</b>
3.1 Capabilities .....	14
3.1.1 Connectivity .....	14
3.1.2 Data Access .....	14
3.1.3 Error Handling .....	14
3.1.4 Robustness .....	14
3.1.5 Dual OPC Interfaces .....	14
3.1.6 Custom Interfaces .....	14
3.1.7 Automation Interfaces .....	14
<b>4 Standards and Conventions .....</b>	<b>15</b>
4.1 OPC Data Access Standards .....	16
<b>5 System Administration Functions .....</b>	<b>17</b>
5.1 Configuration Hierarchy .....	18
5.1.1 HCI/OPC Data Servers .....	18
5.1.2 Windows Event Viewer Configuration .....	18
5.1.3 Timeouts in GUS HCI Client .....	18
5.1.4 NWDDDB .....	18
5.1.5 NWDDDB Server Configuration Settings .....	18
5.2 Installation and removal .....	20
5.2.1 Installation and removal of GUS HCI Client .....	20
5.2.2 Installation and removal of NWDDDB Server .....	20
5.3 Startup and Shutdown .....	21
5.3.1 Startup and Shutdown of GUS HCI Client .....	21
5.3.2 Startup and Shutdown of NWDDDB Server .....	21
5.4 Security Configuration .....	22
5.4.1 Configuration of GUS HCI Client .....	22
5.4.2 DCOM Security .....	22
5.4.3 Configuration of NWDDDB Server .....	22
5.5 Troubleshooting Tools .....	23
5.5.1 General .....	23
5.5.2 Troubleshooting GUS HCI Client .....	23
5.5.3 OPC Device-specific Error Codes .....	29
5.5.4 Troubleshooting NWDDDB Server .....	29
<b>6 Named Data Access Syntax .....</b>	<b>31</b>
6.1 GUS HCI Nameform .....	32
6.2 Scripting Properties .....	33

6.3 Supported Data Types-Named Data Access Syntax .....	35
6.4 Co-Existence of HCI and HOPC Nameforms in the Same GUS Display .....	36
6.5 Programmatic Interfaces - Named Data Access .....	37
6.6 Scenarios and Examples - Named Data Access .....	38
<b>7 Binding and Unbinding HCI/OPC Servers .....</b>	<b>39</b>
7.1 Optimizing the Bind .....	40
7.2 Multiple Binds to the Same Server .....	41
7.3 Multiple Binds to Different Servers .....	42
7.4 Switching Servers .....	43
7.5 Accessing Third-Party OPC servers from HCI .....	44
7.6 UnBinding from a Server .....	45
7.7 Programmatic Interfaces - Binding HCI/OPC Servers .....	46
7.8 Scenarios and Examples - Binding HCI/OPC Servers .....	47
7.8.1 Example 1: Bind to multiple HCI/OPC servers .....	47
7.8.2 Example 2: Bind using an indirect reference .....	47
<b>8 What Does Display Validation Mean to HCI? .....</b>	<b>49</b>
8.1 Summary of GUS HCI client OPC Groups .....	50
<b>9 Synchronous Data Access .....</b>	<b>51</b>
9.1 Programmatic Interfaces-Synchronous Data Access .....	52
9.2 Scenarios and Examples-Synchronous Data Access .....	53
<b>10 Asynchronous Updates .....</b>	<b>55</b>
10.1 Human Interface-Asynchronous Updates .....	56
10.2 Programmatic Interfaces-Asynchronous Updates .....	57
10.3 Scenarios and Examples-Asynchronous Updates .....	58
<b>11 Demand Group Reads .....</b>	<b>59</b>
11.1 Single Group Update .....	60
11.2 All Groups Update .....	61
11.3 FST Update Group .....	62
11.4 Programmatic Interfaces - Demand Group Reads .....	63
11.5 Scenarios and Examples - Demand Group Reads .....	64
11.5.1 Example 1: All data items from the same server .....	64
11.5.2 Example 2: Data items from different servers .....	64
<b>12 Propagation of GUS Keylock to TPN Servers .....</b>	<b>65</b>
12.1 Propagating the security information .....	66
12.1.1 Failure to SetAttributes .....	66
12.2 Programmatic Interfaces - GUS Keylock .....	67
<b>13 Server Failure Detection and Handling .....</b>	<b>69</b>
13.1 Keep-Alive Mechanism .....	70
13.1.1 Detection .....	70
13.1.2 Handling .....	70
13.1.3 Rebinding .....	70
13.2 Programmatic Interfaces – Server Failure .....	71
13.3 Basic Keep-Alive Scenario .....	72
<b>14 Timeouts .....</b>	<b>73</b>
14.1 Bind/UnBind .....	74
14.2 Synchronous Data Access outside of Bind/UnBind Window .....	75
14.3 Asynchronous Scanned Data .....	76
14.4 Keep-Alive Timer .....	77
14.5 Human Interface-Asynchronous Scanned Data .....	78
14.6 Guideline for data access through RDM .....	79

<b>15 Node-Wide Display Database (NWDDDB)</b>	<b>81</b>
15.1 NWDDDB Server and NWDDDB Database	82
15.1.1 Out-of-Proc 'Exe' Server	82
15.1.2 Checkpointing Disabled	82
15.1.3 TPS Managed Component	82
15.1.4 Locked Server	82
15.2 NWDDDB Database Record Format	83
15.3 Server Treatment of Quality	86
15.4 Special Considerations for NAN (Not-a-number)	87
15.5 NWDDDB Database and Package Administration	88
15.6 Error Handling for NWDDDB Database	90
15.7 Accessibility from Applications	91
15.8 Reusability of Displays	92
15.8.1 Scripting Semantics for NWDDDB Server	92
15.8.2 Hierarchical Naming Convention for NWDDDB Data Items	92
15.8.3 Naming Convention for NWDDDB Database	93
15.9 Supported Data Types	94
15.9.1 VARIANT Types Supported by NWDDDB Server	94
15.9.2 Mapping DDB Items to NWDDDB Types	95
15.9.3 Storage of LCN Internal IDs	96
15.9.4 Storage of LCN Type Entity and Type Variable	96
15.10 Support for Indirection	97
15.11 Human Interface	98
15.12 Programmatic Interfaces-NWDDDB Server	99
15.13 Scenarios and Examples - NWDDDB Server	100
15.13.1 Storage of LCN Internal IDs - Scenarios and Examples	100
15.13.2 Storage of LCN Type Entity and Type Variable - Scenarios and Examples	100
15.13.3 NWDDDB Server Startup and Shutdown	102
15.13.4 NWDDDB Server Database Installation w/ Associated Package	102
15.13.5 NWDDDB Database De-Installation w/ Associated Package	102
<b>16 HCI Bind Extensions</b>	<b>105</b>
16.1 Syntax	106
16.2 'Binding' Property	107
16.3 Programmatic Interfaces - Binding Extensions	108
16.4 Scenarios and Examples - Binding Extensions	109
16.4.1 Example 1: Changing the binding	109
16.4.2 Example 2: Binding before and after the 'Bind' statement	110
16.5 Performance Requirements	111
<b>17 Appendices</b>	<b>113</b>
17.1 Appendix A: Comparison of HCI and HOPC data access	114
17.1.1 HOPC & HCI/OPC Data Access Capabilities	114
17.1.2 Consistent Error Handling Across Servers	115
17.1.3 Mapping from LCN Data Types to BasicScript Data Types	115
17.1.4 Display Validation Differences	115
17.1.5 Demand Groups	116
17.1.6 Fast Group Update	116
17.1.7 Indirect references	116
17.2 Appendix B: Default NWDDDB Database	117
<b>18 Notices</b>	<b>119</b>
18.1 Documentation feedback	120
18.2 How to report a security vulnerability	121
18.3 Support	122
18.4 Training classes	123



# 1 About This Document

The GUS HCI Client User's Guide details HCI/OPC client and server data access support for the GUS Display Builder.

The guide describes the syntax and definition of Named Data Access from the GUS Display Builder to OLE for Process Control (OPC) servers, including Honeywell Communications Interface (HCI) servers.

It also discusses the GUS Node-Wide Display Database concept, which is based on a GUS-specific HCI/OPC server.

## Revision history

Revision	Date	Description
A	December 2013	Initial release of document.





## 2 Getting Started

### **Related topics**

“Introduction” on page 10

---

## 2.1 Introduction

The GUS HCI Client User's Guide details HCI/OPC client and server data access support for the GUS Display Builder.

The guide describes the syntax and definition of Named Data Access from the GUS Display Builder to OLE for Process Control (OPC) servers, including Honeywell Communications Interface (HCI) servers.

It also discusses the GUS Node-Wide Display Database concept, which is based on a GUS-specific HCI/OPC server.

### 2.1.1 Assumptions

The reader should be familiar with the GUS Display Builder User's Guide and the Display Scripting User's Guide for information on creating GUS displays. It is assumed that the reader has an adequate understanding of the GUS Display Builder, and also HCI/OPC servers.

This guide will not fully explain how to create displays using the GUS Display Builder, but will only cover those areas that are specifically different for accessing HCI/OPC data.

For more information about HCI/OPC clients and servers, refer to the HCI Client Developer's Guide and the OPC Data Access Specification Reference Manual.

### 2.1.2 Topics

The topics discussed include:

- System Configuration Information
- Syntax specification for Named Data Access, including GUS Display Builder 'Property' extensions
- Binding/Unbinding to/from an HCI/OPC server
- Synchronous Data Access (for example, OnLButtonClick)
- Asynchronous Updates (for example, OnDataChange)
- Demand Group Reads
- Security
- Server Failure Detection
- Timeouts
- Node-Wide Display Database
- HCI Bind Extensions
- Troubleshooting.



#### Attention

The GUS HCI Client can interoperate with HCI/OPC servers as well as with generic OPC servers. Therefore, any unqualified reference to an HCI Server or its data can also be expanded to include a generic OPC server.

---

The purpose of this guide is to provide developers the basic reference material for connecting GUS HCI Client to an HCI or OPC server. It is assumed that the user is familiar with Microsoft COM technology and OPC specifications. It is also assumed that the user is familiar with C++ or Visual Basic.

### 2.1.3 System

OPC and GUS HCI Clients may run on either Windows 7 workstation or server, within a TPS domain or outside a TPS domain.

GUS HCI Client transfers object data between the GUS Display Builder and HCI/OPC or OPC servers.

### Hardware Environment

GUS HCI Client does not introduce additional hardware requirements over those specified by the GUS displays runtime package and particular HCI/OPC data access servers.

### Software Environment

GUS HCI Client does not introduce additional software requirements over those specified by the GUS displays runtime package and particular HCI/OPC data access servers.

### Network Environment

Item	Requirement
PIN/PCN	Ethernet, TCP/IP, Microsoft application layer software such as RPC, DCOM. One of the following name services must be available: DNS WINS Local Host File

## 2.1.4 References

Inside OLE, Second Edition, 1995, Kraig Brockschmidt, Microsoft Press

Component Object Model Specification, Draft Version 0.9, October 24, 1995, Microsoft Corp. and Digital Equipment Corp. (available from Microsoft's FTP site)

OLE for Process Control Specification (OPC), September 11, 1997, Final Version 1.0A

OLE Automation Programming Reference, Microsoft Press, Redmond, WA, 1996

OLE 2 Programming Reference, Vol. 1, Microsoft Press, Redmond, WA, 1994

## 2.1.5 About This Guide

This guide is organized into the following major sections. It also has an extensive glossary for a quick reference for acronyms, terminology, and definitions.

The section entitled 'Functional Overview' provides a description of the capabilities to perform Named Data Access through an HCI or OPC component.

The section entitled 'System Management' describes the configuration required to support the function.

The sections entitled 'Named Data Access' through 'Server Failure Detection and Handling' and 'HCI Bind Extensions' provide information on syntax and key functions provided by the GUS HCI Client.

The section entitled 'Node-Wide Display Database' describes data sharing through a GUS HCI/OPC server.



# 3 Functional Overview

## Related topics

“Capabilities” on page 14

---

## 3.1 Capabilities

The GUS HCI Client with GUS Display Builder provides the user with the capability to perform Named Data Access through an HCI/OPC. Through the GUS Display Builder's Named Data Access, the user is unencumbered from the myriad of HCI, OPC and COM interfaces required to communicate with an HCI/OPC server. The user simply binds to an HCI/OPC server and references a data item through the specified syntax to read or write HCI/OPC server data within a GUS display.

### 3.1.1 Connectivity

Connectivity to multiple HCI/OPC servers within one GUS display, connectivity to the HOPC LCN Data Server along with HCI/OPC server(s), and multiple GUS displays connected to the same HCI/OPC data server are all scenarios supported by GUS Display Builder's Named Data Access.

### 3.1.2 Data Access

Named Data Access supports synchronous reads and writes and asynchronous notifications when data changes. Synchronous reads and writes are done directly to the device. Capabilities are also provided for updating groups of data items at a specified scan rate or on demand based on a group identifier called the Demand Group ID.

### 3.1.3 Error Handling

The common error handling philosophies within the GUS Display Builder apply to HCI/OPC data access. Errors can be handled either by GUS Display Builder's internal error handling or by the user explicitly checking the 'status' parameter and writing an explicit error handler. The GUS HCI Client also provides detailed event logging to the NT Event Viewer Log.

### 3.1.4 Robustness

Over and beyond OPC connectivity, the GUS HCI Client adds robustness. There is detection and indication of a server failure, plus timeout support to ensure that calls to a server do not hang under failure conditions of the server.

### 3.1.5 Dual OPC Interfaces

The OPC Standard provides the specification of two sets of interfaces; the OPC Custom Interfaces and the OPC Automation Interfaces.

### 3.1.6 Custom Interfaces

The GUS HCI Client encapsulates HCI/OPC data access using the OPC Custom Interfaces per the OPC Standard. This means that users can access data from fully compliant HCI/OPC servers, as well as from generic OPC Servers via one common ease of use interface, GUS Display Builder's Named Data Access.

### 3.1.7 Automation Interfaces

Alternately, users can opt to bypass Named Data Access and make the OPC Automation calls directly using the GUS scripts. The GUS Display Builder will undergo product testing to ensure that users can script directly to the OPC Automation Interfaces, apart from the GUS HCI Client. However, this second option of scripting directly to automation calls is somewhat more burdensome for the user in contrast with GUS Named Data Access, which encapsulates the details of COM and OPC from the end user, and also provides robustness.

# 4 Standards and Conventions

## **Related topics**

“OPC Data Access Standards” on page 16

---

## 4.1 OPC Data Access Standards

When communicating with an HCI/OPC data server, GUS HCI Client will follow the methodology outlined by the 'OLE for Process Control' Data Access Custom Interface Standard, Rev 2.04.

If the OPC Data Access server does not implement the Rev 2 of the standard, GUS HCI Client will automatically follow the methodology of Rev 1.0A.



# 5 System Administration Functions

## **Related topics**

“Configuration Hierarchy” on page 18

“Installation and removal” on page 20

“Startup and Shutdown” on page 21

“Security Configuration” on page 22

“Troubleshooting Tools” on page 23

## 5.1 Configuration Hierarchy

There are several configuration areas that are pertinent to the user as explained below.

### 5.1.1 HCI/OPC Data Servers

The HCI and/or OPC data servers are installed and configured independently of the GUS HCI Client. However, GUS HCI Client must be informed about specific data servers that are to be accessed by the GUS displays. The particular server is identified by its name, 'HCI Alias Name'.

Internally, ES-T/Experion system utilities are used for resolving the given server name to:

- location (server host computer)
- DCOM component information (for example, class id of the server object)

### 5.1.2 Windows Event Viewer Configuration

The GUS HCI Client posts error events to the Windows Event Viewer Log. A significant part of each entry is the OPC or COM HRESULT returned internally for a specific method call. This information should be of great benefit to the end user under anomalous circumstances.

Installation / download of GUS HCI Client software automatically configures event log information in the registry.

### 5.1.3 Timeouts in GUS HCI Client

GUS HCI Client timeout/timer values and their corresponding defaults are stored in the registry. All values are of type DWORD. Default values are as indicated in section 13.1, 'Functional Description' of this document. Default values can be changed by the user through the Configuration Utility.

### 5.1.4 NWDDDB

The NWDDDB Server installs as part of the 'GUS Display HCI Client Add-In' package. The NWDDDB server operates within a TPS domain. NWDDDB Server has no user-configurable items.

Installation of NWDDDB Server will write necessary registry information to \HKEY\_LOCAL\_MACHINE\SOFTWARE\Honeywell\MyTpsDomain. NWDDDB Server has no user-configurable items.

### 5.1.5 NWDDDB Server Configuration Settings

NWDDDB Server is a Managed Component, but is not Autostarted by CAS (Component Admin Service). As such, it is visible on the System Status Display. This enables users to stop NWDDDB Server, make updates to their DSS files, and then subsequently restart the server so that NWDDDB Server is reloaded with the updated files.

NWDDDB Server is not configurable through the Configuration Utility since it has no user-definable options.

**Table 1: NWDDDB Server Configuration Settings**

Option	Configured Value	Meaning
'AutoStart'	'No'	Autostarted by CAS (Component Admin Service)
'AuxDisplayProgID'	"	Name of Auxiliary Display on System Status
'BaseProgId'	'Hci.NWDDDB_Server'	Prog ID of server

Option	Configured Value	Meaning
'ConfigDll'	"	Name of DLL if configured by Configuration Utility
'CustomConfigPage'	"	Name of custom configuration page
'HciPersistentFileSpec'	"	Name of checkpointed file
'HCISampleServerDataFile'	'\HWIAC\Checkpoints\ Nwddb\nwddb_server.log'	Pathname\Filename of log file of DSS's
'HostName'	"	Host executing on
'MaxThreadsPerGroup'	10	Number of outstanding asynchronous requests allowed
'OverloadedAsyncAction'	Throttle	Behavior when above limit is exceeded (default is to throttle, and wait for one to free up); other behavior is to return error without waiting.
'TimeThreshHold'	2000	Window of time to wait for the polling thread (based on active group's update rate) to update the cache before forcing another device read to explicitly update cache.
'TPSManaged'	Yes	Server appears in status display and can be started and stopped.

---

## 5.2 Installation and removal

### 5.2.1 Installation and removal of GUS HCI Client

The GUS HCI Client is installed via 'GUS Display HCI Client Add-In' and removed through the Control Panel 'Add/Remove Programs' on the standard ES-T and Remote Displays Nodes.

### 5.2.2 Installation and removal of NWDDB Server

NWDDB Server is an EXE that is installed as part of 'GUS Display HCI Client Add-In' package, and de-installed through Control Panel. Installation of the server requires access to the registry HKEY\_LOCAL\_MACHINE.

---

## 5.3 Startup and Shutdown

### 5.3.1 Startup and Shutdown of GUS HCI Client

The GUS Display Builder automatically loads the GUS HCI Client DLL. There is at most one unique GUS HCI Client per GUS display. The GUS HCI Client exists for the life of its associated GUS Display Builder executable.

If the user is executing a display via the GUS Display Builder, the Client lives until the display is closed. If the user is instead executing displays via GUS Display Builder's runpic option, then the client lives until the GUS Display Builder process is terminated.

### 5.3.2 Startup and Shutdown of NWDDDB Server

NWDDDB Server is launched with the first occurrence of a Bind request after Windows node power-up.

NWDDDB Server runs under the dedicated user account 'LocalComServer'. This account ensures that it remains running across user logins/logouts, and is stopped through Windows node power-down.

Additionally, the NWDDDB Server is available through the System Status Display. This gives the user increased flexibility especially during display development, to stop the NWDDDB Server, make updates to DSS files, and then subsequently reload the new DSS files(s) by restarting the server. Again, the server can be restarted simply by launching a display with a 'Bind' statement to NWDDDB\_Server, or through the System Status Display's 'Start' option.

---

## 5.4 Security Configuration

### 5.4.1 Configuration of GUS HCI Client

GUS HCI Client operates within security context of the currently logged-on Windows user. The HCI method-level data access security is transparent to the GUS HCI Client. An HCI – based OPC Server can grant additional access rights based on identity of the current user.

### 5.4.2 DCOM Security

The GUS display executable process initializes its DCOM security levels at:

- authentication: `RPC_C_AUTHN_LEVEL_NONE`
- impersonation: `RPC_C_LEVEL_IMPERSONATE`.

### 5.4.3 Configuration of NWDDDB Server

There are no specified security requirements for NWDDDB Server, and therefore, it is configured with open security.

## 5.5 Troubleshooting Tools

### Related topics

“General” on page 23

“Troubleshooting GUS HCI Client” on page 23

“OPC Device-specific Error Codes” on page 29

“Troubleshooting NWDDDB Server” on page 29

### 5.5.1 General

Both the GUS HCI Client and the NWDDDB Server log event information to the Windows Event log. This is available through the Viewer with the Source label 'GUSHCIDataAcc', for 'GUS HCI/OPC Data Access'. The detailed error messages will indicate the component reporting the problem (GUSHCIClient, NWDDDB\_Server, and other future components).

### 5.5.2 Troubleshooting GUS HCI Client

The GUS HCI Client cooperates with the GUS Display Builder to provide robust error detection and reporting for the end user. The GUS HCI Client detects error conditions associated with its processing ('HCI CLIENT ERROR'), and reports these in turn to GUS Scripting. GUS Scripting maps the GUS HCI Client-specific error code into a small set of common data access error codes that are made visible to the user via the GUS Scripting Status property, and the Err.Number object. In the table entitled GUS *HCI* Client Errors and GUS Display Builder Scripting Status, these are located under the heading 'GUS DISPLAY BUILDER SCRIPTING STATUS'. This resultant set of scripting statuses is the same set as for HOPC Data Server. An error string is provided in Err.Description. Likewise, these error description strings presented by the GUS Scripting exception mechanism are synonymous between HOPC and HCI data access. For all cases except a write error, the description string is general, and reflects the 'DAS\_\*'-level common error, such as 'Communication Error', and 'Configuration Error', for example. However, for a write error, a more detailed description string is provided. For the Write Error under HOPC data access, the LCN detailed error message is provided along with the error number 1058. A similar level of diagnostic information is made available when the GUS HCI Client determines that there has been a write error. However, unlike HOPCServer data access, the GUS HCI Client does not interrogate device-specific errors; rather, it maps the more detailed 'ppErrors' code into the corresponding LCN error string.

The 'HCI CLIENT ERROR' details are made completely available to the user via an entry to the NT Event Viewer Log.

Error handling via the GUS HCI Client is done at two levels: (1) Errors detected by GUS Scripting, and (2) Errors which are detected by the GUS HCI Client, and then reported to GUS Scripting. Each is discussed in turn.



#### Attention

In some cases a higher-level scripting error may mask the DAS\* error. For example, an attempt to write an invalid type to an OPC data item may be flagged by the higher-level scripting error 13 type mismatch rather than by HCIClient\_SERVER\_PARTIAL\_BADTYPE\_ERROR /DAS\_STORE\_ERROR.

#### Level 1. Errors Detected by GUS Scripting

GUS Scripting detects and reports OLE errors of type DISP\_E\_\*, as indicated in the following table.

**Table 2: OLE Errors Detected by GUS Scripting**

OLE Error	Description	GUS Scripting Error Number
DISP_E_BADPARAMCOUNT	Wrong number of arguments or invalid property assignment	11
DISP_E_BADVARTYPE DISP_E_TYPEMISMATCH	Type mismatch	13
DISP_E_MEMBERNOTFOUND	Property or method not found	438
DISP_E_OVERFLOW	Overflow	6
DISP_E_PARAMNOTOPTIONAL	Argument not optional	449
DISP_E_UNKNOWNNAME	Property or method not found	438

GUS Scripting detects errors associated with the inability to bind to a GUS HCI Client at its level. These are DAS\_COMMUNICATION\_ERROR and DAS\_CONFIGURATION\_ERROR, and are caused by configuration errors.

GUS Scripting may also report DAS\_NO\_ERROR\_BUT\_NO\_DATA for scripts which may be run prior to valid data from the server. An example of this is an OnPeriodicUpdate script.

GUS Scripting may also report an unknown run-time error for undetermined anomalies.

These errors are made available to the user via the GUS Scripting Status attribute, but are NOT reported to the NT Event Viewer Log.

In addition to evaluating the overall status (HRESULT) for each call made to the GUS HCI Client, GUS Scripting as the end-user of client data, also evaluates the OPC Quality for data reads. For quality other than OPC\_QUALITY\_GOOD, the OPC Quality is mapped to the common GUS Display Builder SCRIPTING STATUS to give a consistent indication to the user. This mapping is provided in the following table.

**Table 3: Mapping of OPC Quality to GUS Display Builder Scripting Status**

OPC QUALITY / SubStatus QQSSSSL	HEX Value	GUS Display Builder Scripting Status	ERR NUMBER	VTO CHAR
OPC_QUALITY_GOOD / Non-specific	0xc0	DAS_NO_ERROR	1050	N/A
OPC_QUALITY_GOOD / OPC_QUALITY_LOCAL_ OVERRIDE	0xd8	DAS_NO_ERROR	1050	N/A
OPC_QUALITY_BAD / Non-specific	0x00	DAS_VALUE_ERROR	1053	-
OPC_QUALITY_BAD / OPC_QUALITY_CONFIG_ ERROR	0x04	DAS_CONFIGURATION_ERROR	1052	@
OPC_QUALITY_BAD / OPC_QUALITY_NOT_ CONNECTED	0x08	DAS_NO_ERROR_BUT_NO_DATA	1057	#
OPC_QUALITY_BAD / OPC_QUALITY_DEVICE_ FAILURE	0x0c	DAS_COMMUNICATION_ERROR	1051	?



OPC QUALITY / SubStatus QQSSSSLL	HEX Value	GUS Display Builder Scripting Status	ERR NUMBER	VTO CHAR
OPC_QUALITY_BAD / OPC_QUALITY_SENSOR_ FAILURE	0x10	DAS_COMMUNICATION_ERROR	1051	?
OPC_QUALITY_BAD / OPC_QUALITY_LAST_KNOWN	0x14	DAS_COMMUNICATION_ERROR	1051	?
OPC_QUALITY_BAD / OPC_QUALITY_COMM_ FAILURE	0x18	DAS_COMMUNICATION_ERROR	1051	?
OPC_QUALITY_BAD / OPC_QUALITY_OUT_OF_ SERVICE	0x1c	DAS_CONFIGURATION_ERROR	1052	@
OPC_QUALITY_BAD / Anything else	None of the above	Not applicable. (Unknown Run-time Error)	1059	#
OPC_QUALITY_UNCERTAIN / Non- specific	0x40	DAS_VALUE_ERROR	1053	-
OPC_QUALITY_UNCERTAIN / OPC_QUALITY_LAST_USABLE	0x44	DAS_COMMUNICATION_ERROR	1051	?
OPC_QUALITY_UNCERTAIN / OPC_QUALITY_SENSOR_CAL	0x50	DAS_VALUE_ERROR	1053	-
OPC_QUALITY_UNCERTAIN / OPC_QUALITY_EGU_ EXCEEDED	0x54	DAS_VALUE_ERROR	1053	-
OPC_QUALITY_UNCERTAIN / OPC_QUALITY_SUB_NORMAL	0x58	DAS_COMMUNICATION_ERROR	1051	?
OPC_QUALITY_UNCERTAIN / Anything else	None of the above	Not applicable. (Unknown Run-time Error)	1059	#
NOT OPC_QUALITY_GOOD and NOT OPC_QUALITY_BAD and NOT OPC_QUALITY_UNCERTAIN	!0xc0 !0x00 !0x40	Not applicable. (Unknown Run-time Error)	1059	#

### Level 2. Errors Detected by the GUS HCI Client

Errors detected by the GUS HCI Client, (except for HCIClient\_UUID\_FAILURE which is reported from the client constructor and HCIClient\_INTERNAL\_EXCEPTION), are propagated to GUS Scripting for exposure to the user through the GUS Scripting Status attribute. Additionally, an entry is posted by the GUS HCI Client to the NT Event Viewer Log to provide further information to the user for diagnosing the cause of the problem. The actual OPC HRESULT is not directly made available to the user via a scripting property; however, it is part of the log entry. This entry minimally contains the OPC HRESULT or Microsoft HRESULT (if the request failed in Microsoft COM code as opposed to an OPC call). Additional details, such as the server name, group name, item name, and so on, are also provided in the log entry to further qualify the nature of the problem. Note that in the case of an HCIClient\_SERVER\_SPECIFIC\_PARTIAL\_ERROR, the HRESULT logged is the HRESULT which has been extracted from 'ppErrors', per the OPC Specification. Also note that if the user opts not to write an error handler, that error information is still available via the log entry.

At its level, the GUS HCI Client deals primarily with two types of errors: (a) those associated with invalid user inputs to scripts, and (b) those errors which are more internal to the Client processing. The first type of errors is primarily associated with 'invalid' parameters in scripts that are eventually propagated to the GUS HCI Client (for example, invalid server namestring, undefined data reference, undefined demand group ID). For these errors, it is particularly appropriate for the user to write error handlers, and to take appropriate action based on the returned status.

Other errors are more internal to the GUS HCI Client processing in that they are not the direct effect of a user input, but still denote a failure of the system to function as the user intended. Examples of these are the Client's inability to create a group at the OPC Server, or a failure experienced when requesting memory. These run-time errors will similarly be logged to the NT Event Viewer, and be propagated back to the user as GUS Scripting Status. The entry log will contain the call's HRESULT plus the origin of the error (C++ method name) within the client, and any other pertinent information which can help the user to diagnose the problem.

Table 4: GUS HCI Client Errors and GUS Display Builder Scripting Status

HCI CLIENT ERROR / GUS Display Builder SCRIPTING STATUS	EXPLANATION	ERR NUM- BER	VTO CHAR
HCIClient_UUID_WARNING / Not applicable.	GUS_HCIClient detected inability to create UUID during client construction.	N/A	N/A
HCIClient_IOMap_SetAttrErrPolicy_ERROR	GUS_HCIClient detected inability to set Permissive Error Policy for IOMap Server.	N/A	N/A
HCIClient_SERVER_PARTIAL_CLAMP_ERROR DAS_NO_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; OPC_S_CLAMP Error.	1050 or 1058	N/A
HCIClient_SERVER_PARTIAL_S_OK_MISMATCH DAS_NO_ERROR or DAS_WRITEERROR if detected during write operation.	GUS_HCIClient detected partial server error but S_OK Mismatch returned for individual item.	1050 or 1058	N/A
Not applicable. / DAS_COMMUNICATION_ERROR	HCI bind cannot complete; check registry entry /GUS Display Builder/ config1.	1051	?
HCIClient_BIND_ERROR / DAS_COMMUNICATION_ERROR	GUS_HCIClient detected bind error for input server name.	1051	?
HCIClient_SERVER_ERROR / DAS_COMMUNICATION_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected server error; E_FAIL/Unknown Status returned from server function.	1051 or 1058	?
HCIClient_SERVER_PARTIAL_E_FAIL_ERROR / DAS_COMMUNICATION_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; E_FAIL Error returned for individual item.	1051 or 1058	?
HCIClient_SERVER_GET_STATUS_ERROR / DAS_COMMUNICATION_ERROR	GUS_HCIClient detected server in FAILED/NOCONFIG/ SUSPENDED/TEST State.	1051	?
HCIClient_TIMEOUT_ERROR / DAS_COMMUNICATION_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected timeout for call to server.	1051 or 1058	?
Not applicable. / DAS_CONFIGURATION_ERROR	HCI bind cannot complete; check registry entry /GUS Display Builder/ DataProviders/HCI.  NOTE: This error is currently being masked by a higher-level GUS Display Builder error, and this is being addressed.	1052	@

HCI CLIENT ERROR / GUS Display Builder SCRIPTING STATUS	EXPLANATION	ERR NUM- BER	VTO CHAR
HCIClient_DUPLICATENAME_ERROR / DAS_CONFIGURATION_ERROR	GUS_HCIClient detected duplicate name input for AddGroup call to server.	1052	@
HCIClient_INVALID_GROUP_ERROR / DAS_CONFIGURATION_ERROR	GUS_HCIClient detected invalid demand group ID for server.	1052	@
HCIClient_INVALIDARG_ERROR / DAS_CONFIGURATION_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected invalid argument passed to server function.	1052 or 1058	@
HCIClient_ONDATACHANGE_ERROR / DAS_CONFIGURATION_ERROR	GUS_HCIClient failed to establish callback connection to server . May require 2-way trust relationship. May be caused by the inability of the server to impersonate the client.	1052	@
HCIClient_PUBLIC_ERROR / DAS_CONFIGURATION_ERROR	GUS_HCIClient detected attempt to add items to a public group for server.	1052	@
HCIClient_UNSUPPORTEDRATE_ERROR / DAS_CONFIGURATION_ERROR	GUS_HCIClient detected server unable to provide requested update rate, but errantly offered faster rate.	1052	@
HCIClient_UNSUPPORTEDRATE_INFO / DAS_NO_ERROR	GUS_HCIClient detected server unable to provide requested update rate, and has offered slower rate.	N/A	N/A
HCIClient_OUTOFMEMORY_ERROR / DAS_CONFIGURATION_ERROR	GUS_HCIClient failed to allocate memory internally.	1052	@
HCIClient_SERVER_OUTOFMEMORY_ERROR / DAS_CONFIGURATION_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected insufficient memory to perform requested server function.	1052 or 1058	@
HCIClient_SERVER_PARTIAL_BADRIGHTS_ERROR / DAS_CONFIGURATION_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; OPC_E_BADRIGHTS Error returned for individual item.	1052 or 1058	@
HCIClient_SERVER_PARTIAL_UNKNOWNITEMID_ERROR / DAS_CONFIGURATION_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; OPC_E_UNKNOWNITEMID Error returned for individual item.	1052 or 1058	@
HCIClient_SERVER_PARTIAL_UNKNOWNPATH_ERROR / DAS_CONFIGURATION_ERROR	GUS_HCIClient detected partial server error; OPC_E_UNKNOWNPATH Error returned for individual item.	1052	@
HCIClient_SERVER_PARTIAL_INVALIDITEMID_ERROR / DAS_CONFIGURATION_ERROR	GUS_HCIClient detected partial server error; OPC_E_INVALIDITEMID Error returned for individual item.	1052	@
HCIClient_SERVER_PARTIAL_ACCESSDENIED_ERROR / DAS_CONFIGURATION_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; E_ACCESSDENIED returned for individual item.	1052 or 1058	@

HCI CLIENT ERROR / GUS Display Builder SCRIPTING STATUS	EXPLANATION	ERR NUM- BER	VTO CHAR
HCIClient_SERVER_PARTIAL_RANGE_ERROR / DAS_VALUE_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; OPC_E_RANGE Error returned for individual item.	1053 or 1058	-
HCIClient_NAN_ERROR / DAS_VALUE_ERROR	GUS_HCIClient detected NAN for float/double.	1053	-
HCIClient_SERVER_PARTIAL_BADTYPE_ERROR / DAS_STORE_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; OPC_E_BADTYPE Error returned for individual item.	1053 or 1058	-
HCIClient_INTERNAL_ERROR / DAS_STORE_ERROR	GUS_HCIClient detected internal software error.	1054	!
HCIClient_SERVER_PARTIAL_INVALIDHANDLE_ERRO R / DAS_STORE_ERROR or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; OPC_E_INVALIDIHANDLE Error returned for individual item.	1054 or 1058	!
HCIClient_NULL_POINTER_ERROR / DAS_STORE_ERROR	GUS_HCIClient detected NULL Pointer error.	1054	!
HCIClient_NOINTERFACE_ERROR / DAS_NOT_YET_SUPPORTED	GUS_HCIClient detected interface NOT supported for server function.	1055	#
HCIClient_VT_ARRAY_UNSUPPORTED_ERROR / DAS_NOT_YET_SUPPORTED	GUS_HCIClient detected attempted use of unsupported type VT_ARRAY.	1055	#
HCIClient_SERVER_PARTIAL_NOTIMPL_ERROR / DAS_NOT_YET_SUPPORTED	GUS_HCIClient detected partial server error; E_NOTIMPL returned for individual item.	1055	#
HCIClient_SERVER_SPECIFIC_PARTIAL_ERROR / Not applicable. (Unknown Run-time Error) or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected partial server error; S_FALSE returned from server function.	1055 or 1058	#
HCIClient_SERVER_SPECIFIC_ERROR / Not applicable. (Unknown Run-time Error) or DAS_WRITEERROR if detected during write operation	GUS_HCIClient detected server- specific error.	1055 or 1058	#
Not applicable. / DAS_NO_ERROR_BUT_NO_DATA	Display start-up occurrence, where no data has arrived from the server yet but a script is run. Typically seen in OnPeriodicUpdate scripts.	1057	#
Set of all GUS HCI Client errors associated with doing a synchronous write / DAS_WRITEERROR	All HCIClient errors associated with Doing a synchronous write regardless of specific reason.	1058	N/A
Not applicable. / Not applicable. (Unknown Run-time Error) or DAS_WRITEERROR if detected during write operation	Detection of unknown data access error.	1059 or 1058	#

### 5.5.3 OPC Device-specific Error Codes

If a given HCI/OPC server adds vendor-specific errors in 'ppErrors' for a function for which this is legal, such as IOPCSyncIO::Read/Write, the GUS HCI Client will commonly treat the status as HCIClient\_SERVER\_SPECIFIC\_PARTIAL\_ERROR. In this manner, the user is notified that an 'unknown' error specific to the server has occurred. The detailed information for the error, including the HRESULT returned for the specific OPC call is available to the user through the NT Event Viewer.

### 5.5.4 Troubleshooting NWDDB Server

The NWDDB Server will be visible on the Experion System Status Display. This enables users to view its operational status, as well as to stop and to restart the server.

There is error detection and reporting for the NWDDB Database, and this is detailed in the following paragraphs.

#### NWDDB Database

The NWDDB Server's log file, NWDDB\_Server.log tells NWDDB Server what .dss Input Files to load. Error checking is done on the log file contents to check for comments, duplicate files and non-existing files. If the log file is empty, non-existent, and/or no indicated .dss file can be successfully loaded, then NWDDB Server makes one last attempt to load the Default NWDDB Database, nwddb\_server.dss. Errors associated both with the .dss and log file are logged to the NT Event Viewer as per the Table below *GUS NWDDB Errors*.

Both the default .dss file and the log file are copied to C:\ProgramData\Honeywell\Checkpoints\Nwddb\ at installation time. Note that if the user already has a log file at this destination, it is not overwritten by a re-installation of the server.

Users have been advised to use hierarchical naming conventions to avoid name collisions. Duplicate checks are made on OPC data items as they are being loaded into NWDDB Server's memory and a warning, NWDDB\_SERVER\_DUPLICATE\_ITEM\_ERROR is written to the NT Event Log. No data items are discarded by NWDDB. The responsibility for sorting out discrepancies is left to the user.

Errors associated with accessing the NWDDB Database are detected and reported by GUS Scripting in conjunction with the GUS HCI Client.

Errors in the .dss file associated with missing/incorrect data items will result in a Configuration Error of type OPC\_E\_UNKNOWNITEMID when a user tries to access a data item that is not present in NWDDB Server's Database. The user is notified via a scripting error, and also has additional information in the NT Event Log.

Mismatches in data type between the defaulted initial value and the type indicated by the Vt Type Tag result in an error at the GUS Scripting level of Error 13/Type Mismatch.

If the Default NWDDB Database is subsequently corrupted or missing, these are also manifested as Configuration Errors when the user attempts to access a data item of interest.

**Table 5: GUS NWDDB Errors**

NWDDB Error	EXPLANATION
NWDDB_SERVER_DSS_LOAD_ERROR	NWDDB_Server detected inability to load requested DSS File.
NWDDB_SERVER_LOG_FILE_ERROR	NWDDB_Server detected inability to locate or utilize LOG File.
NWDDB_SERVER_DUPLICATE_ITEM_ERROR	NWDDB_Server detected duplicate item name across DSS Files = N and item still added.
NWDDB_SERVER_DUPLICATE_FILE_ERROR	NWDDB_Server detected duplicate .DSS file to load.



## 6 Named Data Access Syntax

The GUS HCI Client with GUS Display Builder provides the user with a simple and unencumbered syntax for accessing HCI/OPC data. However, it does nothing to preclude a user from scripting directly to the OPC Automation Interfaces.

Since there are multiple HCI/OPC data servers, a GUS HCI Nameform must specify the associated data server. A 'Bind' method is provided which enables a user to correlate a specific HCI server name to a server variable. In this way, if the user wishes to subsequently redirect data to another HCI/OPC server, this can be easily accomplished simply by updating the server name provided to the Bind method. All other HCI/OPC data references in GUS scripts remain intact.

# 6.1 GUS HCI Nameform

The GUS HCI Nameform used in HCI Named Data Access is as follows:

hci.<serveralias>{.<namealias>}.<namealias>	
where:	
<serveralias> :	a symbolic name/alias representing an HCI/OPC Server internally in scripts.
<namealias> :	either a symbolic name/alias if HCI Bind Extensions is used; or a name if not. This can include a point name, a parameter name, or a combination of the above.



## 6.2 Scripting Properties

Specification of a property in an HCI nameform follows the 'parameter\_name' portion in the name syntax; that is., 'point\_name.parameter\_name.property'. If no property is explicitly included, the default is to the 'Value' property.

The properties that are scriptable by the user are:

1.	Value is the value of the point_name.parameter_name. Access to the value will generate a run-time error if the status is not DAS_NO_ERROR.
2.	ValueNoError also specifies the value but will NOT automatically generate a run-time error if the status is other than DAS_NO_ERROR. It is left to the user to check the Status property after the read for correctness.
3.	TimeStamp is the time and date that the data item was last read.
4.	Quality represents the quality state for an item's data value. The low eight bits of the Quality flags are currently defined in the form of three bit fields: Quality, Sub-status, and Limit Status, and are arranged as follows:

QQSSSSLL, where the Quality (QQ) bit fields are as follows (information is per the *OPC Data Access Specification Reference Manual, OPC Data Access Custom Interface Standard* in the 'Description of Data Types, Parameters, and Structures,' 'OPC Quality Flags' section:

**Table 6: OPC Standard Quality Bit Fields**

Quality (QQ) Bits	Define	Description
0x00	Bad	Value is not useful for reasons indicated by the sub-status.
0x01	Uncertain	The quality of the value is uncertain for reasons indicated by the sub-status.
0x10	Not applicable	Not used by OPC.
0x11	Good	The quality of the value is good.

The OPC Standard also supplies the following Comments about Quality:

- A server that supports no quality information must return Good. It is also acceptable for a server to simply return Bad or Good and to always return zero for sub-status and limit.
- Even when a 'BAD' value is indicated, the contents of the value field must still be a well-defined VARIANT although it is not accurate so as to simplify error handling in client applications.
- Even if the server has no known value to return, then some reasonable default should be returned such as a NULL string or a zero numeric value.

The Sub-status and Limit Status bits are detailed in the OPC Standard.

1.	<b>Type</b> is the integer data type as defined by the server of the data item specified.
2.	<b>Status</b> is the HCI returned status of the data item last read. The Status property is used for checking the correctness of the value just read. It can also be used for user-written error handlers as demonstrated in subsequent sections. The user can obtain more detailed information for those errors detected by the GUS HCI Client in the NT Event Viewer. Also, please note that HCIClient_UUID_FAILURE does NOT get returned as a scripting status, but is only logged to the NT Event Viewer Log. This is because it is generated from the GUS HCI Client's constructor, which cannot be a returned status to GUS Scripting.
3.	Status can contain one of the values enumerated in the HCI Client Errors and GUS Display Builder Scripting Status table.
4.	<b>Binding</b> The 'Binding' property retrieves the actual value for a portion of the GUS HCI nameform that is indirected. For more information, refer to the 'HCI Bind Extensions' section of this guide.



**Attention**

- If a nameform includes a property keyword as its name, it must be escaped through a square bracket as follows:
  - Nameform = testmode.value, escaped name = hci.s1.testmode.[value]
  - Where s1 is the server alias. Users are discouraged from embedding properties into nameforms that they define, as for NWDDB Server.
-

## 6.3 Supported Data Types-Named Data Access Syntax

The GUS HCI Client provides support for all HCI/OPC Data Types, which are all types that are based on VARIANT. However, when the GUS HCI Client runs in its native environment, as a DLL loaded by the GUS Display Builder, the set of types usable by both the GUS HCI Client and automation clients of the GUS Display Builder is limited by the data types that are provided by the Display Builder's scripting language, BasicScript.

GUS HCI Client requests data in canonical format, that is, as stored internally at the server. On receiving data, the particular VARIANT type is coerced into the expected BasicScript type.

BasicScript Data Type	VARIANT Data Type
Boolean	VT_BOOL
Currency	VT_CY
Date	VT_DATE
Double	VT_R8
Integer	VT_I2
Long	VT_I4
Object	VT_DISPATCH
Single	VT_R4
String	VT_BSTR
Variant	VT_VARIANT
String	VT_BSTR

### Indexed data items

Elements of arrays are retrieved individually, with current index value.

For all distinct indexed data items in the GUS scripts, separate OPC data items will be declared in the OPC collection groups. On change of index, affected OPC collection group will be updated.

Example:

Two onDataChange scripts, both accessing data items from the **ArrayItems** array:

```
Sub onDataChange
Text1.text = Hci.s.ArrayItems(I)
End Sub
Sub onDataChange
Text2.text = Hci.s.ArrayItems(J)
End Sub
```

The OPC collection group(s) will contain two items:

ArrayItems(*Current Value of I*)

ArrayItems(*Current Value of J*)

---

## 6.4 Co-Existence of HCI and HOPC Nameforms in the Same GUS Display

The GUS HCI Client provides the capability for the user to indicate HCI as well as HOPC nameforms in the same GUS display.

Recall that a HOPC nameform is indicated with the prefix of 'lcn', and does not contain the server specification. Therefore, it is of the form lcn.point.parameter.

---

## 6.5 Programmatic Interfaces - Named Data Access

To access HCI/OPC data items through a GUS Display the user must specify 'hci' as the root, the HCI/OPC server name, the HCI/OPC data item (or point.parameter), and optionally the GUS Display Builder available properties.

The syntax is as follows:

```
hci.<server_variable_name>.<point_name.parameter_name>[.<property>]
```

Where

hci is the fixed, root name used to communicate with an HCI/OPC compatible server.

server\_variable\_name is the internal scripting name assigned in the HCI Bind statement specifying the server with which to communicate.

point\_name.parameter\_name is the name of an HCI/OPC data item on the specified server.

Optional property. For more information, refer to the “Scripting Properties” on page 33 section of this guide.

## 6.6 Scenarios and Examples - Named Data Access

Following is an example of a GUS Display Builder script that reads the value of an HCI/OPC data item 'p101.pv' from HCI/OPC server with HCI Alias Name 'HCIDataServer1' and displays the value in the display.

```
Sub OnButtonClick()
  hci.server1.bind 'HCIDataServer1'
  me.text = hci.server1.p101.pv
End Sub
```

When the user clicks the left mouse button down, the GUS HCI Client will first bind to the HCI/OPC data server 'HCIDataServer1' and then read the item 'p101.pv' from that server. In this case, if an error occurs on the bind or the read, the GUS Display Builder will automatically generate an exception error message box.

The user can optionally write custom error handlers for exceptions as shown in the following write example. In this case the user is overriding the built-in exception error message box and handling the error in whatever manner is necessary.

```
Sub OnButtonClick()
  On Error Goto Error_Handler
  hci.myserver.bind 'HCIDataServer1'
  hci.myserver.p101.pv = hci.myserver.p101.pv + 1
  hci.myserver.p102.pv = hci.myserver.p102.pv + 10
Exit Sub
Error_Handler:
  select case Err.Number
  case DAS_COMMUNICATION_ERROR:
    MsgBox 'Error Occurred in Bind. Error Number is ' + CStr(Err.Number)
  case else
    MsgBox Err.Description + '. Error Number is ' + CStr(Err.Number)
  end select
End Sub
```

To avoid automatic exception handling of errors, the user may opt to access the value using the ValueNoError property. This allows the user to ignore access errors entirely, avoiding the need for error handling. Or the user can access the Status property and handle the error as shown below.

```
Sub OnButtonClick()
  hci.s1.bind 'Template_Server1'
  me.text = hci.s1.a101.pv.ValueNoError
  DIM stat
  stat = hci.s1.a101.pv.Status
  if stat <> HCI_NO_ERROR then
    Goto Error_Handler
  end if
Exit Sub
Error_Handler:
  select case stat
  case DAS_CONFIGURATION_ERROR
    MsgBox 'Read reported configuration error.'
  case DAS_VALUE_ERROR
    MsgBox 'Read reported value error.'
  case else
    MsgBox 'Error number is ' + CStr(stat)
  end select
End Sub
```

## 7 Binding and Unbinding HCI/OPC Servers

Before any HCI/OPC data access from a GUS display can occur, the user must bind to an available HCI/OPC server. The bind creates a connection from the display to the specified server. Servers are referenced in a nameform using the `server_variable_name` specified in the GUS Display Builder script's Bind syntax. If an error occurs during the bind, an exception is automatically generated. The user can then choose to let the GUS Display Builder run-time display an exception dialog or handle the exception using an error handler.

### **Related topics**

- “Optimizing the Bind” on page 40
- “Multiple Binds to the Same Server” on page 41
- “Multiple Binds to Different Servers” on page 42
- “Switching Servers” on page 43
- “Accessing Third-Party OPC servers from HCI” on page 44
- “UnBinding from a Server” on page 45
- “Programmatic Interfaces - Binding HCI/OPC Servers” on page 46
- “Scenarios and Examples - Binding HCI/OPC Servers” on page 47

---

## 7.1 Optimizing the Bind

For a given display, the 'Bind'script only needs to appear only once for each unique HCI/OPC server of interest. The OnDisplayStartup script, or a user event script are recommended vehicles for the 'Bind'method.



---

## 7.2 Multiple Binds to the Same Server

A single display can have multiple connections to the same HCI/OPC server.

Normally, the user would specify unique `server_variable_name(s)` for each unique server to be bound. The effect of binding to the same server name via the same `server_variable_name` results in an internal request to unbind from the given server, and then the new Bind request is made. Since the Bind operation is expensive, this type of scripting behavior is strongly discouraged.

---

## 7.3 Multiple Binds to Different Servers

A single display can have multiple binds to different HCI/OPC servers. The user would simply specify a Bind request to each of the servers of interest via their associated server names, along with a unique `server_variable_name` for each.

---

## 7.4 Switching Servers

One of the benefits of the Named Data Access syntax is that it readily enables a user to specify an alternate data source. This can be accomplished simply by re-scripting the server name indicated in the 'Bind' script via the Display Builder. Then all nameforms, which include the `server_variable_name` bound to this new server, will access data from the new server. If the HCI/OPC servers being used are pre-known, then the display author can set up event scripts in a display that can rebind to an alternate server, with no changes required to the original .pct file. If the HCI/OPC servers of interest are all pre-known, and the display author decides at a later time to use an alternate HCI/OPC server, then just a single line needs to be altered in the original .pct file, that is, the Bind request.

At run-time, clicking on a script which specifies a new server namestring for an existing `server_variable_name` will result in an unbind from the current HCI/OPC server, and a bind to the newly-specified server. Since the scripting symbol table is re-walked at bind time, all data references in the display are carried across to the newly-bound server.

---

## 7.5 Accessing Third-Party OPC servers from HCI

The HCI server names used in a GUS display can be defined as a Third Party OPC Server. You must establish a name for the Third Party OPC server prior to use within a display. To establish an HCI Name, add the Third Party OPC server to your TPS Domain using the TPS Domain Configuration utility. Follow the steps below to access a Third Party OPC server from HCI.

- 1 Install the server on the desired Experion node per the vendor's instructions. (Note: The server's client installation package must be installed on every ES-T or Experion APP Node that needs to access it.
- 2 On a node where you have installed the Third Party OPC server, pull up the HCI Component Configuration page in the TPS domain configuration utility.
- 3 Select the 'Configure Installed Component...' button.
- 4 Select the progID for the Third Party OPC server.
- 5 Enter a unique name in the component name field.
- 6 Enter the host ID of the Experion node where you installed the server.
- 7 Select 'Check Name' button.
- 8 If no error is reported, select OK.
- 9 Replicate.

Now the Third-Party OPC Server can be accessed by GUS displays in exactly the same way that GUS Displays would access an HCI Server. However, the Third Party OPC Server is not managed by the Experion system and will therefore not appear on the Experion system Status Display.

---

## 7.6 UnBinding from a Server

Disconnect from a server can be either explicitly requested by the user, or it can occur implicitly when a display is closed or the communication to the server is lost. To request an UnBind from a given server, the user enters either a NULL server name, or omits the server name altogether in the 'Bind' method.



### Attention

- There is no explicit UnBind method; it is achieved by the absence of a namestring parameter or by a NULL namestring parameter in the Bind method. Throughout the remainder of this document, any reference to an unbind should be understood to be implemented through the Bind method.

When the UnBind activity is complete, the user will be provided one DAS\_COMMUNICATION\_ERROR for all active scan references, and continuous DAS\_COMMUNICATION\_ERROR's for all data items in an OnPeriodicUpdate script at the interval of invocation. Any future attempt to access data items through user event scripts will also give the DAS\_COMMUNICATION\_ERROR.

Any VTO's in a display whose 'Value' tab indicates a data reference that is associated with an unbound server will show the 'question mark' character.

Display objects that use scripts to access data values will show the last valid value until the user explicitly re-binds to the associated server. Therefore, in scripts associated with objects that display process values accessed via the GUS HCI Nameform, it is recommended that the user handle the DAS\_COMMUNICATION\_ERROR's associated with unbinding be handled with an 'On Error' handler that displays an indication that the data is no longer valid, similar to the VTO.

GUS display will not automatically rebind servers with which it has lost communication. The display needs to be re-invoked or the user can opt to rebind to a given server through a user event script that contains the Bind method. Then, data access updates for On-Data-Change will resume, and the user can again request synchronous data accesses through scripting events.

Whenever a display is closed by a user, implicit UnBinds are done for all connected data servers.

---

## 7.7 Programmatic Interfaces - Binding HCI/OPC Servers

To bind to an HCI/OPC server from a GUS display the following scripting syntax is used. Note that this script only needs to appear once in a display, such as in an OnDisplayStartup script or user event script.

```
hci.<server_variable_name>.bind <HCI_server_name>
```

Where

- hci is the fixed, root name used to communicate with an HCI/OPC compatible server.
- server\_variable\_name is the variable name used in all subsequent data access references to the given HCI\_server\_name.
- HCI\_server\_name must resolve to the HCI Alias Name for a valid HCI/OPC server. If NULL, or if this parameter is NOT present, an Unbind will result. Note that the following syntax can be used to represent an alias in another TPSDomain: '//TPSDomainName/ComponentName'.



### Attention

- If you are accessing the HCI/OPC server on a different TPS domain, ensure to use the name of the resident TPS domain.
-

## 7.8 Scenarios and Examples - Binding HCI/OPC Servers

### 7.8.1 Example 1: Bind to multiple HCI/OPC servers

Following is a GUS Display Builder script that completes a bind to multiple HCI/OPC servers with writes to items from each server. Any exceptions raised from the binds or writes will be handled by the error handler.

```
Sub OnButtonClick()
On Error Goto Error_Handler
hci.myserver1.bind 'HCIDataServer1'
hci.myserver2.bind 'HCIDataServer2'
hci.myserver1.p101.pv = hci.myserver1.p101.pv + 1
hci.myserver2.p102.pv = hci.myserver2.p102.pv + 10
Exit Sub
Error_Handler:
select case Err.Number
case HCI_COMMUNICATION_ERROR
Msgbox 'Error Occurred in Bind. Error Number is ' + CStr(Err.Number)
case else
Msgbox Err.Description + '. Error Number is ' + CStr(Err.Number)
end select
End Sub
To unbind from myserver1, a user would enter either of the two following scripts:
Sub OnButtonClick()
hci.myserver1.bind ''
End Sub
Sub OnButtonClick()
hci.myserver1.unbind
End Sub
```

### 7.8.2 Example 2: Bind using an indirect reference

This example illustrates a bind that utilizes an indirect reference that resolves to a server names.

```
Sub OnDisplayStartup()
hci.nwddb.bind 'NWddb_Server'
msgbox 'Bound to NWddb Server'
hci.s1.bind hci.nwddb.Server1
msgbox 'Bound to a server whose HCI Alias Name is stored in the nwddb string data item 'Server1''
End Sub
```





## 8 What Does Display Validation Mean to HCI?

The act of validating a display containing data references results in the creation of the GUS Scripting Symbol Table. This is a tree containing all the components of all the nameforms contained within a given display.

Users bind to HCI/OPC servers via the GUS HCI Client at run-time via scripts. Once the client has been successfully bound to an HCI/OPC server, GUS Scripting creates the 'OnDataChange' OPC collection groups for all items in OnDataChange scripts and in OnPeriodicUpdate scripts. Errors are reported during this process for failures associated with creating groups and successfully adding their respective items.

After the 'binding' process, there exists a unique OnDataChange OPC Group for each unique update rate and Demand Group ID specified by the user.

Note additionally, that the GUS HCI Client creates at most one 'Immediate Group' for all items that are activated by the user through event scripts. Since at display run-time, the user may opt not to activate any of these scripts, this group is created with the very first activation of an event script, such as OnLButtonClick, that contains a data item in either a read/write access. Also note that these data items are added to this OPC Group whenever they are first referenced.

---

## 8.1 Summary of GUS HCI client OPC Groups

The GUS HCI Client associated with each unique display creates the following OPC Groups depending on the type of data accesses requested for the data items contained in a display. Maximally, the following OPC Groups can be created for a given HCI/OPC server that is accessed from the GUS display:

1.	One OPC Group for all items involved in Synchronous data accesses (both reads and writes) \ 'Immediate Group' (May be empty if user never selects any data items via user event scripts.)
2.	One onDataChange OPC Group for each unique combination of requested update rate and Demand Group ID (Reference GUS Display Builder's Data Collection pull-down)

## 9 Synchronous Data Access

Named Data Access supports synchronous reads and writes of HCI/OPC data items. Synchronous reads and writes are done via device accesses. Synchronous reads and writes are typically activated by overt user actions (OnLButtonClick), which are also called 'user event scripts'. These data accesses do not return control until the GUS HCI Client completes its communication with the HCI/OPC server. Error handling can be handled automatically by the GUS Display Builder-Run-time exception handler, or the user may write a custom error handler. For examples of user-written error handlers, refer to the 'Named Data Access Syntax' section of this guide.

---

## 9.1 Programmatic Interfaces-Synchronous Data Access

For synchronous Data accesses, there are no additional scripting interfaces added. The user simply accesses data within the standard user interface scripts, such as OnLbuttonClick.

---

## 9.2 Scenarios and Examples-Synchronous Data Access

The following example shows a synchronous read of an HCI/OPC data item, 'p101.pv', from an HCI/OPC server named 'HCIDataServer1'. The server is first bound to the server variable name 'server1'. The value is then stored into a text object to be displayed on the screen.

```
Sub OnButtonClick()  
hci.server1.bind 'HCIDataServer1'  
me.text = hci.server1.p101.pv  
End Sub  
Following is a write example where HCI/OPC data item 'p101.pv' is incremented by 1.  
Sub OnButtonClick()  
hci.myserver1.bind 'HCIDataServer1'  
hci.myserver1.p101.pv = hci.myserver1.p101.pv + 1  
End Sub
```



# 10 Asynchronous Updates

Named Data Access supports asynchronous data access, which happens independent of overt user events. These accesses are accomplished through GUS Scripting's internal cache. The user can set up asynchronous updates through an OnDataChange script, through an OnPeriodicUpdate script, or by directly specifying a GUS HCI nameform in the Value tab of a VTO. Note that when the value associated with a point.parameter changes, all references to that data item in a given display are updated with the new cached value.

Subroutines in a display object scripted with the name 'OnDataChange' update items only when an item's value changes. The Display/Data Collection/Rate (sec) pulldown on the GUS Display Builder enables a user to indicate the requested rate at which the server is to poll the device for a new value. Effectively, this is the fastest rate at which the GUS HCI Client may receive OnDataChange callbacks. Note that the server may elect to execute data updates at a slower rate.

Subroutines in a display object scripted with the name 'OnPeriodicUpdate' execute every half-second. If 'OnPeriodicUpdate' scripts contain data items, these are from the internal cache.

The intended use for the OnPeriodicUpdate entry point is to support simple types of display animation. The OnDataChange entry point is recommended over the OnPeriodicUpdate entry point for dynamic displays. For more information, refer to the *GUS Scripting Guide*.

## Related topics

“Human Interface-Asynchronous Updates” on page 56

“Programmatic Interfaces-Asynchronous Updates” on page 57

“Scenarios and Examples-Asynchronous Updates” on page 58

---

## 10.1 Human Interface-Asynchronous Updates

The GUS Display Builder is the human interface that is being afforded this new functionality.

Requested update rate is specified using the GUS Display Builder's Display/Data Collection/Rate (sec) pulldown. Note that in GUS Display Builder update rate = 0 indicates single read (update once). OPC standard does not support the 'read-once' explicitly so that a requested update rate of zero is treated by the GUS HCI Client as a request for a read once per day (24 hours). This results in the desired behavior of the single initial read, with a subsequent read a very long duration later. This is acceptable since this rate is typically used just to fetch initial values of strings that are not expected to change.

By using the GUS Display Builder's VTO (for example, the Value tab in text object properties page), an asynchronous update of an HCI/OPC data item can be obtained.

The user simply specifies the expression as a GUS HCI Nameform. When the value changes, an internal script is executed to display the item.



---

## 10.2 Programmatic Interfaces-Asynchronous Updates

For asynchronous Data accesses, there are no additional scripting interfaces added. The user simply uses the standard OnDataChange and OnPeriodicUpdate scripts.

---

## 10.3 Scenarios and Examples-Asynchronous Updates

The following example shows an asynchronous update of an HCI/OPC data item, 'p101.pv'. This example assumes a valid HCI/OPC server is first bound to the server variable name 'server1'. When the item's value changes, the value is stored into GUS Display Builder's internal cache. All scripts referencing the cached item are then queued to run. As each script runs, the value is taken from cache. As with synchronous reads, the user can choose to allow automatic exception handling of errors, write their own error handler, or chose to access items using the ValueNoError property to ignore exceptions. This example shows the use of a user-written error handler.

```
Sub onDataChange()  
On Error Goto Error_Handler  
me.text = hci.server1.p101.pv  
Exit Sub  
Error_Handler:  
select case Err.Number  
case DAS_COMMUNICATION_ERROR  
Msgbox 'onDataChange reported communication error for item.'  
case DAS_CONFIGURATION_ERROR  
Msgbox 'onDataChange reported communication error for item.'  
case else  
Msgbox Err.Description + '. Error Number is ' + CStr(Err.Number)  
end select  
End Sub
```

# 11 Demand Group Reads

To support immediate refresh of a group of HCI/OPC data items, the GUS Display Builder enables the user to group data items by Demand Group ID. The Demand Group ID is specified through GUS Display Builder's Display/Data Collection/Demand Group ID pulldown.

Points from different HCI/OPC servers can be grouped together by specifying the same Group ID. Since OPC groups are specific to a given HCI/OPC server, the GUS HCI Client creates unique groups, potentially with the same Demand Group ID, on different servers. To force an immediate refresh of a group of data items, the user enters a script containing the 'Update' method. Note that the 'Update' method is a method on the GUS HCI Client object, rather than a method on a given server, since demand group membership can span different OPC servers.

The demand group refresh forces the server to perform immediate update of the particular OPC collection group with data source 'from device'.

## **Related topics**

“Single Group Update” on page 60

“All Groups Update” on page 61

“FST Update Group” on page 62

“Programmatic Interfaces - Demand Group Reads” on page 63

“Scenarios and Examples - Demand Group Reads” on page 64

---

## 11.1 Single Group Update

GUS Named Data Access will refresh a group of items by specifying the Demand Group ID of interest.

---

## 11.2 All Groups Update

The user can request an immediate refresh across all Demand Groups by specifying Group ID zero.

---

## 11.3 FST Update Group

The Fast Update group is denoted by 'FST' by the GUS Display Builder in the Display/Data Collection/Demand Group ID pulldown.

All data items that are present in the FST group will be updated at the server's fastest rate available if the 'Fast Update' toggle is active in the running GUS display.

Note: internally, all affected OPC collection groups will switch collection rates to 'collect as fast as possible'.

Activation and de-activation of the FST group in a running GUS display is achieved through:

- gus display runtime pop-up menu item 'Fast Update'
- IKB 'Fast' key

In a standard GUS runtime environment, where displays may access both HOPC and HCI/OPC data, the HOPC data server limits number of displays that can activate 'Fast Update' to 1.

In a Desktop Experion environment there is no limit on number of displays that may have 'Fast Update' concurrently active.

---

## 11.4 Programmatic Interfaces - Demand Group Reads

The syntax for specifying a demand group read is as follows:

hci.Update <DemandGroupID>

Where

- the DemandGroupID has been pre-established via the Display/Data Collection/Demand Group ID pulldown on the GUS Display Builder Window.

Note that the server\_variable\_name is NOT part of the Update method. Conceptually, the Update request is being made of the GUS HCI Client for all of its connected servers in a running display.

## 11.5 Scenarios and Examples - Demand Group Reads

### 11.5.1 Example 1: All data items from the same server

Demand Group update of Single Group (all data items from the same server)

Assumptions:

- The user has previously bound to an HCI/OPC server s1.
- The user has specified via the GUS Display Builder pull-down Display/Data Collection that hci.s1.p101.pv and hci.s1.p102.pv both have Demand Group ID 1.

hci.Update 1

- Will provide an immediate update for both hci.s1.101.pv and hci.s1.102.pv.

### 11.5.2 Example 2: Data items from different servers

Demand Group update of Single Group (data items from different servers)

Assumptions:

- The user has previously bound to an HCI/OPC server s1.
- The user has previously bound to an HCI/OPC server s2.
- The user has specified via the GUS Display Builder pull-down Display/Data Collection that hci.s1.p101.pv and hci.s2.p102.pv both have Demand Group ID 2.

hci.Update 2

- Will provide an immediate update for both hci.s1.101.pv and hci.s2.102.pv.

#### Example 3

Demand Group update of all Groups

Assumptions:

- The user has previously bound to an HCI/OPC server s1.
- The user has previously bound to an HCI/OPC server s2.
- The user has specified through the GUS Display Builder pull-down Display/Data Collection the following:
  - hci.s1.p101.pv has Demand Group ID 1.
  - hci.s2.p102.pv and hci.s2.p103.pv both have Demand Group ID 2.

hci.Update 0

- Will provide an immediate update for all values.



## 12 Propagation of GUS Keylock to TPN Servers

### **Related topics**

“Propagating the security information” on page 66

“Programmatic Interfaces - GUS Keylock” on page 67

---

## 12.1 Propagating the security information

The GUS HCI Client is required to detect the current keylock position for an ES-T node, and to propagate this security information to TPN HCI/OPC servers. The keylock applies for all data access write requests made by the HCI TPN Server on behalf of the GUS Client.

Whenever a user makes a Bind request to a server, the GUS HCI Client uses the HCI-specific 'GetAttributes' method to validate that a given server supports the internal property called 'AccessLevel'. A return of S\_OK indicates that the server supports this property, and that keylock propagation processing should commence.

The 'SetAttributes' method allows the GUS HCI Client to apply current keylock position (presented as 'AccessLevel') on this client-server connection. An event mechanism is also enabled at this time to allow the GUS HCI Client to detect any subsequent changes in the keylock position, and to forward these to the server.

Potential sources of keylock are:

- the TPS Infrastructure Signon Manger (using COP), including IKB and the SetKeylock utilities
- GUS display scripting keylock actors

### 12.1.1 Failure to SetAttributes

The HCI TPN server may refuse change of the 'AccessLevel' based on insufficient credentials of the currently logged-on (Windows logon) user at ES-T. If the 'SetAttributes - Access' command fails, the last previously accepted value will stay in effect.

Insufficient keylock access level issues are manifested on a GUS display by exceptions/VTO character indicators at the time that a write occurs and fails.

---

## 12.2 Programmatic Interfaces - GUS Keylock

There are no new user visible interfaces used to support setting the AccessLevel attribute on a server. Keylock propagation is performed automatically on Bind request and anytime ES-T key level changes.



# 13 Server Failure Detection and Handling

## **Related topics**

“Keep-Alive Mechanism” on page 70

“Programmatic Interfaces – Server Failure” on page 71

“Basic Keep-Alive Scenario” on page 72

---

## 13.1 Keep-Alive Mechanism

Once the GUS HCI Client has successfully bound to an HCI/OPC server, a pinging mechanism is established between itself and each connected server.

### 13.1.1 Detection

Via the GetStatus OPC method, the GUS HCI Client requests the server's status at a regular interval. Lack of response within timeout or any status other than 'OPC\_STATUS\_RUNNING' or 'OPC\_STATUS\_TEST' is interpreted as a server failure.

### 13.1.2 Handling

GUS HCI Client automatically UnBinds from the failed server, thus generating communication errors to the scripts

### 13.1.3 Rebinding

Once a failed server comes back online, the user needs to explicitly rebind the server in order to be able to communicate with this server. Rebind operation may be initiated via a specific user interface 'Bind' in a script or by re-invoking the display - if server is bound at display startup.

---

## 13.2 Programmatic Interfaces – Server Failure

No new programmatic interfaces are added for Server Failure Detection and Handling. It is an internal function.

---

## 13.3 Basic Keep-Alive Scenario

- Create a display with an explicit Bind method in an OnLButtonClick script, plus scanned data access.
- Run the display and bind to an HCL/OPC server of interest.
- Observe data values on the display.
- From Windows NT Task Manager/Processes, select the server process, and click End Process.
- After approximately the interval of the Keep-Alive Timer has expired, observe scripting errors or '?' for scanned data values.
- Perform the OnLButtonClick to rebind to the server.
- Observe the return of data values on the display.
- Reconfigure the value of the Keep-Alive Timer to various values, repeat the 'Basic Keep-Alive Scenario', and observe the difference in the detection interval.



# 14 Timeouts

The following paragraphs detail the technical approach used by the GUS HCI Client to implement timeouts. The in-going assumption was that no single call to an HCI/OPC server could cause a GUS display to hang without the ability to notify the user of the problem, and without the ability to close the display. Whenever possible, the HCI Timeouts facility was used to keep the approach consistent with the TPS Infrastructure recommendations.

The GUS HCI Client had to solve the problem of timeout protection across all modes of GUS communication with an HCI/OPC server. This includes binding/unbinding, synchronous reads and writes and asynchronous data change callbacks.

The GUS HCI Client provides rich timeout protection against the potential for a call made from the client to hang if an HCI/OPC server fails midstream while processing the call. That is, the server would have to be healthy when receiving the call, and then fail while executing the call. If a timeout occurs, the user will be advised via a COMMUNICATION ERROR at the scripting level, and a timeout error with the specifics of the failure will be entered in the NT Event Viewer. Depending on the nature of the failure, the user can resolve the anomaly at the server end, and then either re-requests the operation (for example, UnBind and Re-Bind server) without closing the display, or chose to close the display. New timed threads will be launched for retried operations. Moreover, any threads in a hung state will be explicitly terminated as a result of either unbinding from a server, or closing the display. The key benefit is that the display will not hang even though a low-level call to the server may hang.

---

## 14.1 Bind/UnBind

The GUS HCI Client's *Bind* and *UnBind* operations are composed of numerous COM and OPC calls.

The Bind operation includes the COM call CoCreateInstance, which has the potential to hang at the server under failure for a significant amount of time.

UnBind is composed primarily of removing OPC groups, and doing releases on COM objects. These operations and the myriad of COM/OPC calls involved were protected from timeout by designing UnBind as an asynchronous operation. From a GUS scripting point of view, an unbind request gives an immediate return to the client user, but is asynchronous in the sense that its associated cleanup activities on the server complete later. The UnBind timeout value applies only to the asynchronous portion.

There is a user-configurable timeout value for Bind with default value.

The UnBind timeout is fixed at 60 seconds, and does not affect execution of GUS displays.

If the client detects failure to complete a Bind request within the specified interval, an HCICLIENT\_TIMEOUT\_ERROR will be generated, which the user will ultimately see as a COMMUNICATION ERROR.

Following a Bind timeout, the user can subsequently adjust the configured timeout value, or take other corrective action and then retry the operation without closing the display.

---

## 14.2 Synchronous Data Access outside of Bind/UnBind Window

The remaining synchronous calls made by the Client which fall outside of the Bind/UnBind windows are a combination of COM and OPC calls. All OPC /COM calls are covered by the explicit use of the HCI timed interface version for these calls.

All synchronous data access calls that are outside of the Bind/UnBind window are timed at the user-configurable interval called 'Access' timeout, which is defaulted.

---

## 14.3 Asynchronous Scanned Data

Quiescent Time, will be defined as the period outside of the Bind/UnBind windows, with only scanned data access ongoing. This means no synchronous requests are being made, and the GUS HCI Client is in the state of waiting for callbacks associated with onDataChange items.

When the GUS HCI Client is waiting for callbacks directly from an underlying server, the KeepAlive mechanism, as detailed under the 'Server Failure Detection and Handling' section, comes into play. It provides protection against a server failing, versus just a state when no data is changing, and provides a user-visible indication of server failure in a GUS display.

---

## 14.4 Keep-Alive Timer

For more details about the GUS HCI Client's Keep-alive mechanism, refer to the 'Server Failure Detection and Handling' section of this guide.

The GUS HCI Client provides a user-configurable TIMER interval for the frequency at which keep-alive pings are issued to all HCI/OPC servers. The default value is provided, and this interval applies commonly for all connected servers.

On its GetStatus request to each HCI/OPC server, which is the ping, the GUS HCI Client uses the timed Infrastructure method, such that a failure of the GetStatus call itself will not hang the keep-alive mechanism (that is, the keep-alive itself is timed). The GetStatus timeout value is internally set to 35 seconds to allow for the occurrence of an RPC timeout with an RPC error return. This has been demonstrated through testing to be 33 plus seconds for the remote case.

## 14.5 Human Interface-Asynchronous Scanned Data

The GUS HCI Client timeout values and timer values are configurable through the GUS HCI Client Timeouts Configuration Page shown below.

The **Bind** timeout indicates amount of time spent on waiting for successful connection to the HCI/OPC server (execution of the scripting Bind method).

Minimal value:	5000 msec
Maximum value:	600000 msec
Initialized to:	20000 msec

The **Ping** interval indicates rate of polling of a connected HCI/OPC server. A poll is executed using a read method that is governed by the **Access** timeout. Each server is being polled independently. If a response to a poll message is not received within **Access** time then server failure is assumed.

Minimal value:	5000 msec
Maximum value:	600000 msec
Initialized to:	30000 msec

Note: The maximum time to detect loss of communication with a server is equal to a combined value of Ping and Access intervals.

The **Access** timeout indicates maximum amount of time spent on waiting for response for any method executed on a connected HCI/OPC server. That includes:

- immediate read and write operations
- all internal operations on a connected server (for example, change of update rate for FastUpdate groups)
- ping operations

Minimal value:	5000 msec
Maximum value:	600000 msec
Initialized to:	10000 msec

The configuration page validates the limits and announces errors on non-valid entries.

---

## 14.6 Guideline for data access through RDM

If using GUS displays to connect to redundant servers (such as Experion Servers) through Redirection Manager (RDM), you need to increase the setting for the HCI Client Access Timeout to 30 seconds. The current 10-second default is designed for direct connections to standard HCI Servers - it does not allow enough time for failover when connecting through the RDM, so a longer Access Timeout is needed.

Minimal value:	30000 msec
Maximum value:	600000 msec
Initialized to:	10000 msec

**CAUTION**

Increasing the Access timeout substantially would have users wait an inordinant amount of time before communications errors were reported.

---





# 15 Node-Wide Display Database (NWDDDB)

Node-Wide Display Database provides a solution to HCI/OPC users for storing and accessing data across multiple displays, that is, multiple GUS Display Builder processes within the same node, but not across multiple nodes.

This is an HCI solution. The approach requires the installation and use of the GUS Display HCI Client Add-In Package along with the other GUS Display packages of interest.

Node-Wide Display Database is not meant to be an immediate replacement to the GUS Display Database (DDBs). Rather, NWDDDB will be a phased solution where in initial stages, there is still a reliance on DDBs to help facilitate some TPN functionality.

## **Related topics**

“NWDDDB Server and NWDDDB Database” on page 82

“NWDDDB Database Record Format” on page 83

“Server Treatment of Quality” on page 86

“Special Considerations for NAN (Not-a-number)” on page 87

“NWDDDB Database and Package Administration” on page 88

“Error Handling for NWDDDB Database” on page 90

“Accessibility from Applications” on page 91

“Reusability of Displays” on page 92

“Supported Data Types” on page 94

“Support for Indirection” on page 97

“Human Interface” on page 98

“Programmatic Interfaces-NWDDDB Server” on page 99

“Scenarios and Examples - NWDDDB Server” on page 100

## 15.1 NWDDDB Server and NWDDDB Database

The NWDDDB solution is based on a new HCI/OPC Server called the NWDDDB Server, that reads and 'writes' data to a persistent file in the same manner as HCI's test server, `template_server1`. This file is commonly known as the server 'Input File', and is characterized by its '.dss' extension.

The NWDDDB\_Server must run on a node that has the 'System Management Runtime' package installed. For its configuration and packaging aspects, refer to the '“System Administration Functions” on page 17' section of this guide.

### 15.1.1 Out-of-Proc 'Exe' Server

Each NWDDDB Server has a co-resident flat textual file with default name '`nwddb_server.dss`' that is used to populate the server's cache with data items and set initial conditions, such as value each time the server is restarted. This Input File of defaults is also referred to as the 'Default NWDDDB Database' for the remainder of this writing. Once cached, the 'NWDDDB Database' is node-global in that it can be accessed by multiple HCI/OPC clients within a node. In the GUS Display Builder environment, there is exactly one GUS HCI Client instance per GUS Display Builder process executing a display. Therefore, since this new server will be used to access one common database used by multiple GUS Display Builder processes each running a given display, it is designed as an HCI out-of-process 'EXE' server. Note that if a given data item's value changes during the execution of a display, that new value is ONLY recorded in RAM, but not recorded back to the Input File. The rationale is that an updated value is only meaningful for a given instant in time; therefore, updated values should not persist once the server is shut down. However, the updated values in the server's cache are used across various executions of GUS displays as long as the NWDDDB Server is not shut down and restarted.

### 15.1.2 Checkpointing Disabled

Since the NWDDDB Server will interface with a database for storage and retrieval of local data items rather than points located on an external device, checkpointing of internal IDs to an '.hci' file has been disabled. The database of initial values is located in `C:\ProgramData\Honeywell\Checkpoints\Nwddb\nwddb_server.dss`. Note that at startup time, NWDDDB Server's cache will be populated with the contents of this initial Default NWDDDB Database, plus all Input Files annotated in its log file. For more information, refer to the 'NWDDDB Database and Package Administration' section of this guide.

### 15.1.3 TPS Managed Component

NWDDDB Server is a TPS Managed component. As such, its operational status is available through the System Status Display, and it can be selectively stopped and restarted by the user. However, it is not defined with the Autostart option. This means that it is not started nor stopped by CAS (Component Admin Service). Instead, it is launched either by the GUS Display Builder with the first occurrence of a 'Bind' statement after node restart, or through the System Status Display 'Start' option.

### 15.1.4 Locked Server

The NWDDDB Server is created at the first occurrence of a Bind request after NT power-up. Then, it is locked to ensure that it stays running even if there are currently no bound clients. The NWDDDB Server is not Autostarted because startup is a low overhead event ( $\leq 1.0$  seconds) that typically occurs only once per node reboot.

## 15.2 NWDDDB Database Record Format

At installation time, the NWDDDB Database will be populated with Honeywell defaults as provided in the Appendix entitled: 'Default NWDDDB Database'. There are data items for each of the supported data types. There is one additional data item 'GUS.CWDDDB\_Item', that represents a placeholder for console-wide communication.

Each field in a given row must be separated by one or more space and/or TAB characters.

Comments must be designated by a double-slash (//) delimiter.

The 'GUS.CWDDDB\_Item' is shown below to explain the record structure of the NWDDDB Database.

**Table 7: NWDDDB Database Record Format**

Data Item	Vt Type Tag Value	Quality	Initial Value
...			
GUS.CWDDDB_Item	3	0xc0	100
...			

The Data Item name is left to the user, but should include the package name and be hierarchical in nature, as described in a subsequent section called 'Hierarchical Naming Convention for NWDDDB Data Items'.

The Vt Type Tag is the numerical value for the variant type tag, and should be taken from the table under 'Supported Data Types/ VARIANT Types Supported by NWDDDB Server'.

The OPC Quality/Substatus/Limit bits total eight, and are as follows: QQSSSSLL. '0xc0' represents good quality, and is recommended as the default quality setting. The list of quality settings is provided in the table below.

**Table 8: OPC Quality Settings**

Quality Indicator	HEX	DEFINE	TREATMENT	WHO
OPC_QUALITY_GOOD / Non-specific	0xc0	The value is good. There are no special conditions.	Generated if user provides good value for NAN or for uninitialized data item at run-time.	NWDDDB
OPC_QUALITY_GOOD / OPC_QUALITY_ LOCAL_OVERRIDE	0xd8	The value has been Overridden. Typically, this means the input has been disconnected and a manually entered value has been 'forced'.	Not applicable; no external device connections.	N/A
OPC_QUALITY_BAD / Non-specific	0x00	The value is bad but no specific reason is known.	Generated for NAN. Generated for OPC_E_ BADTYPE	NWDDDB GOPC
OPC_QUALITY_BAD / OPC_QUALITY_ CONFIG_ERROR	0x04	There is some server-specific problem with the configuration. For example, the item in question has been deleted from the configuration.	Generated for OPC_E_ UNKNOWNITEMID if item was there and then deleted.	GOPC

Quality Indicator	HEX	DEFINE	TREATMENT	WHO
OPC_QUALITY_BAD / OPC_QUALITY_ NOT_CONNECTED	0x08	The input is required to be logically connected to something but is not. This quality may reflect that no value is available at this time, for reasons like the value may have not been provided by the data source.	Generated if user does not provide an initial value for a data item; set to GOOD when user provides a value at run-time.  GOPC receives HCI_E_CACHEPOOR from GetData request.	NWDDDB GOPC
OPC_QUALITY_BAD / OPC_QUALITY_ DEVICE_FAILURE	0x0c	A device failure has been detected.	Not applicable; no external device connections.	N/A
OPC_QUALITY_BAD / OPC_QUALITY_ SENSOR_FAILURE	0x10	A sensor failure has been detected.	Not applicable; no external device connections.	N/A
OPC_QUALITY_BAD / OPC_QUALITY_ LAST_KNOWN	0x14	Communications have failed. However, the last known value is available. Note that the 'age' of the value may be determined from the TIMESTAMP in the OPCITEMSTATE.	Not applicable; no external device connections.	N/A
OPC_QUALITY_BAD / OPC_QUALITY_ COMM_FAILURE	0x18	Communications have failed. There is no last known value available.	Generated for failed Read.	GOPC
OPC_QUALITY_BAD / OPC_QUALITY_ OUT_OF_SERVICE	0x1c	The block is off scan or otherwise locked. This quality is also used when the active state of the item or the group containing the item is InActive.	Generated when group and/or data item active state indicates 'InActive'.	GOPC
OPC_QUALITY_ UNCERTAIN / Non-specific	0x40	There is no specific reason why the value is uncertain.	Default. The initial value of quality prior to successfully accessing the value.	GUS Scripting
OPC_QUALITY_ UNCERTAIN / OPC_QUALITY_ LAST_USABLE	0x44	Whatever was writing this value has stopped doing so. This error is associated with the failure of some external source to 'put' something into the value within an acceptable period of time.	Not applicable; no external device connections.	N/A
OPC_QUALITY_ UNCERTAIN / OPC_QUALITY_ SENSOR_CAL	0x50	Either the value has been 'pegged' at one of the sensor limits (in which case the limit field should be set to 1 or 2), or the sensor is otherwise known to be out of calibration via some form of internal diagnostics (in which case the limit field should be 0).	Not applicable; no external device connections.	N/A
OPC_QUALITY_ UNCERTAIN / OPC_QUALITY_ EGU_EXCEEDED	0x54	The returned value is outside the limits defined for this parameter. Note that in this case the 'Limits' field indicates which limit has been exceeded but does NOT necessarily imply that the value cannot move farther out-of-range.	Not applicable; limits not implemented.	N/A

Quality Indicator	HEX	DEFINE	TREATMENT	WHO
OPC_QUALITY_ UNCERTAIN / OPC_QUALITY_ SUB_NORMAL	0x58	The value is derived from multiple sources and has less than the required number of Good sources.	Not applicable; single source of data.	N/A

Finally, the user assigns an initial value for the data item. For assistance in formatting the initial value for each supported data type, refer to Appendix A.

---

## 15.3 Server Treatment of Quality

“Table 8: OPC Quality Settings” contains a column called 'DEFINE' and another called 'TREATMENT'. 'DEFINE' is the OPC definition per the section OPC Quality Flags. 'TREATMENT' describes the circumstances used by the server to generate this quality setting. Through the Input DSS File, an initial value for quality can be indicated. This value holds until a circumstance arises by the server to change this setting. A given quality setting can be generated either by the NWDDB Device-Specific server portion or by GOPC. In one instance, it is generated by GUS Scripting. This is also distinguished by the 'WHO' column.

For further information on OPC Quality, and corresponding GUS Scripting errors at the user level, reference “Table 2: OLE Errors Detected by GUS Scripting”:

---

## 15.4 Special Considerations for NAN (Not-a-number)

At runtime, users can initialize NWDDDB Server data item of Type VT\_R4 to a NAN value by an assignment to a string of three or more dashes as follows:

```
Hci.nwddb.r4 = '---'
```

Reading a data item with a NAN value will result in a DAS\_VALUE\_ERROR, because the NWDDDB Server treats a NAN as an item with OPC\_QUALITY\_BAD.

Subsequently setting the data item to a valid value will return the quality to OPC\_QUALITY\_GOOD, and clear the error.

## 15.5 NWDDDB Database and Package Administration

Package authors who wish to take advantage of NWDDDB are required to take additional steps when they release their packages.

First, and foremost, the NWDDDB Database Input File that is being released with a package must be renamed to some meaningful name representative of that software package. However, the '.dss' suffix must be retained. An example for a NWDDDB Database name for ACME Chocolate Bar displays could be 'ACMEChocolateBarsWithNuts.dss'.

The NWDDDB Server will maintain a log file called 'nwddb\_server.log' in C:\ProgramData\Honeywell\Checkpoints\Nwddb. This log file contains the list of all NWDDDB Database Input Filenames with path extensions, which are to be used to populate the NWDDDB Server's cache at startup. It has one initial entry, which is the Default NWDDDB Database, C:\ProgramData\Honeywell\Checkpoints\ nwddb\_server.dss. Each time a package is installed that relies on NWDDDB, it must make an entry in NWDDDB Server's log file. Each time a package is de-installed, the package's install scripts must locate its associated log file entry and delete it from the log file. An example entry in the log file is C:\ProgramData\Honeywell\Checkpoints \ ACMEChocolateBarsWithNuts.dss.

A package's install scripts must copy any associated NWDDDB Database Input Files to a new subdirectory of NTFS. A new subdirectory of the root C:\ProgramData\Honeywell\Checkpoints \NWDDDB is recommended; for example, C:\ProgramData\Honeywell\Checkpoints \NWDDDB\RAC. Along with the copy, the scripts must also append an entry to NWDDDB Server's log file.

When NWDDDB Server begins executing, it will iterate through each entry in its log file, locate the indicated NWDDDB Database Input File, and perform an 'Open/Append' operation until a new concatenated file has been created and loaded into memory. The new file contains the union of all information contained in the Default NWDDDB Database plus all information across all NWDDDB Database Input Files for all packages that have registered with the NWDDDB Server's log. This complete concatenation is the 'NWDDDB Database'. If no additional entries are made in the log, then NWDDDB Server runs off just the default nwddb\_server.dss file.

A package's de-install scripts are required to delete any associated NWDDDB Database Input Files from the appropriate install directory, and likewise, to remove the filename entry from the log file.



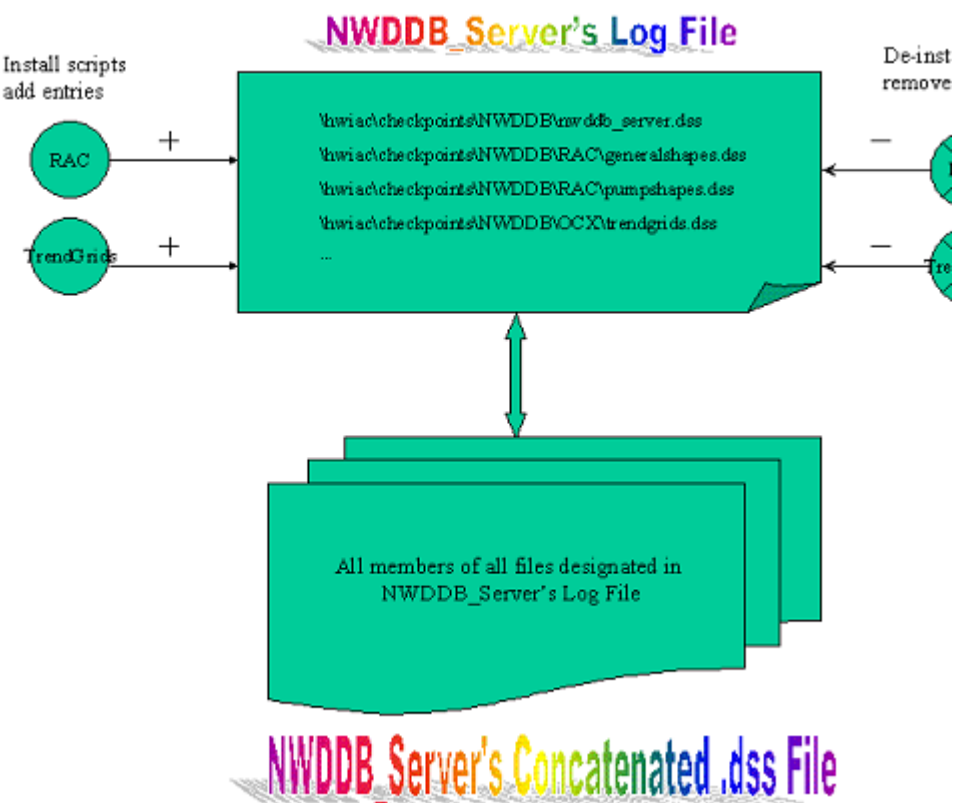


Figure 1: NWDDB Server's Concatenated Database

---

## 15.6 Error Handling for NWDDDB Database

For more information, refer to the '“NWDDDB Database”' section of this guide.

---

## 15.7 Accessibility from Applications

The NWDDB Server is HCI-based, and as such supports both the OPC custom and automation interfaces. This aspect enables connectivity from either a C++ client, or from an application that supports OLE Automation, such as VB, VC++, and VBA-based applications. The NWDDB Server loads the GOPC DLL, and this DLL exports a type library called 'OPCDisp', which provides access to the OPC automation interfaces.

## 15.8 Reusability of Displays

### 15.8.1 Scripting Semantics for NWDDDB Server

In order that displays can be reused without modification, each client node must talk to a local instance of the NWDDDB Server that has a well-known name and has a co-resident persistent data file. Note that the well-known registered name for this server is '**NWDDDB\_Server**', and this name will be the same on every installed node.

'NWDDDB\_Server' will be the server namestring used in the HCI 'Bind' statement. It is also required that users adopt the common server variable name '**nwddb**' to associate the server namestring in the Bind. This will be more performant than using various aliases, since multiple rebinds will not occur, for example, in embedded displays. (That is, if a bind has already been done to 'NWDDDB\_Server' using alias nwddb, subsequent Bind calls using the same semantics will simply cause a return, and not proceed with an additional bind. If various aliases are used rather than the common alias nwddb, each will result in a new unique bind request to the NWDDDB Server.)

Users should not explicitly unbind from the NWDDDB Server. It is especially important to maintain this connection, for example, when users are working in embedded displays. By default, whenever the main display is closed, all server connections are terminated.



#### Attention

To summarize, the following naming conventions are required for the NWDDDB Server, where the bolded words are significant:

#### Sub OnDisplayStartup

```
hci.nwddb.bind 'NWDDDB_Server'
```

```
End Sub
```

### 15.8.2 Hierarchical Naming Convention for NWDDDB Data Items

Display authors are encouraged to use a hierarchical naming convention when defining NWDDDB data items in their Input Files. This will allow users in separate locations, and with no knowledge of each other to define DDB variables with a reasonable assurance that the names do not conflict with one another.

It is recommended that users adopt a hierarchical name structure for their NWDDDB OPC data items, thus minimizing the possibility of name conflicts.

For example, consider a display author in Anytown, USA who is in the process of constructing a suite of Embedded Displays (called 'Paper Mill Objects') for reuse as a component in GUS displays. The Embedded Displays in this suite read and write to a NWDDDB data item named 'FP\_Color'. The Anytown author defines these data items under a directory named after his suite of components ... for example, a script in this suite may read as follows:

```
Sub OnDataChange
me.text = hci.nwddb.PaperMillObjects.FP_Color
End Sub
```

The Anytown author then releases with his suite of objects a text file called 'PaperMillObjects.dss' defining all the NWDDDB data items that the suite assumes. Since the names of the NWDDDB data items are preceded by the name of the suite, namely, 'PaperMillObjects', the chances of some other GUS display object using the same name is minimized, allowing maximum interoperability between separately developed packages.

A user at ACME Chocolate Company is developing wrappers for a new chocolate bar. He wants to use the 'Paper Mill Objects' embedded display in his own display, which features the new chocolate bar. To do this, he simply obtains the 'Paper Mill Objects' display, along with its associated NWDDDB Database. Using a text editor, he then appends the 'borrowed' 'PaperMillObjects.dss' NWDDDB Database to his own NWDDDB Database. Then, the Anytown data item appears in his NWDDDB Database as follows, along with a second 'FP\_Color' data item

that is specific to his chocolate bar display. This fragment of the ACME NWDDB Database called 'ACMEObjects.dss' is presented below.

**Table 9: NWDDB Database File Fragment**

Data Item	Vt Type Tag Value	Initial Quality	Initial Value
ACMEObjects.FP_Color	3	0xc0	80
PaperMillObjects.FP_Color	3	0xc0	80

### 15.8.3 Naming Convention for NWDDB Database



#### **Attention**

Once a user is ready to release a software package, the user must rename the Default NWDDB Database filename prefix 'nwddb\_server' to a meaningful name that matches his display suite. However, the '.dss' extension must remain. The renamed NWDDB Database Input File and its corresponding suite of displays should be released together.

## 15.9 Supported Data Types

Since NWDDDB is facilitated through HCI/OPC Communication, the realm of Supported Data Types is based on the VARIANT as discussed in a previous section in this document entitled 'Supported Data Types'. The GUS HCI Client is limited in the data types that it can request to those data types that are supported by the GUS Display Builder scripting language, Basic Script. The subsection 'VARIANT Types Supported by NWDDDB Server' provides a full list of VARIANT types supported by the NWDDDB Server, even though not all these are requestable under Basic Script. If the GUS HCI Client is instead loaded by a C++ application rather than GUS Scripting, or if a C++ client other than GUS HCI Client is used, for example, all the data types that the NWDDDB Server offers can be accessed.

The intent is that NWDDDB become the HCI functional equivalent [and more] of the GUS Display Database (DispDB), allowing that different GUS scripting may be needed to accommodate certain functions. This includes indirection support for both entities and variable types as supported under the current Display Database. This mapping is discussed in the section entitled 'Mapping DDB Items to NWDDDB Types'.

There is also a section dedicated to storage of LCN Internal IDs for use in Changezone displays.

### 15.9.1 VARIANT Types Supported by NWDDDB Server

The following table indicates the VARIANT Supported Data Types by the NWDDDB Server.

Table 10: VARIANT Type Mappings for NWDDDB Server

VARIANT Data Type	Vt Type Tag Value	Supported by NWDDDB Server	Basic Script Data Type
VT_UI1	17	X	
VT_I1	16	X	
VT_I2	2	X	Integer
VT_I4	3	X	Long
VT_R4	4	X	Single
VT_R8	5	X	Double
VT_BOOL	11	X	Boolean
VT_ERROR	10	X	
VT_CY	6	X	Currency
VT_DATE	7	X	Date
VT_BSTR	8	X	String
VT_UNKNOWN			
VT_DISPATCH			Object
VT_ARRAY   *	8192	Array of VARIANT of Type: VT_I1, VT_UI1, VT_I2, VT_I4, VT_CY, VT_R4, VT_R8, VT_BOOL, VT_ERROR, VT_DATE, VT_BSTR  Arrays of: VT_R4, VT_R8, VT_I1, VT_I2, VT_I4, VT_UI1, VT_BSTR	
VT_BYREF   VT_UI1			

VARIANT Data Type	Vt Type Tag Value	Supported by NWDDDB Server	Basic Script Data Type
VT_BYREF   VT_I2			
VT_BYREF   VT_I4			
VT_BYREF   VT_R4			
VT_BYREF   VT_R8			
VT_BYREF   VT_BOOL			
VT_BYREF   VT_SCODE			
VT_BYREF   VT_CY			
VT_BYREF   VT_DATE			
VT_BYREF   VT_BSTR			
VT_BYREF   VT_UNKNOWN			
VT_BYREF   VT_BSTR			
VT_ARRAY   *			
VT_BYREF   VT_VARIANT			
Generic ByRef			

### 15.9.2 Mapping DDB Items to NWDDDB Types

The following table maps the available properties of the GUS Display Database object to their corresponding NWDDDB Variant data type. Note that NWDDDB makes no distinction between local and global storage. All data items are globally accessible across displays. Also, note that there is no sense of blocks of reserved data items in the NWDDDB Database as there is for DDBs. The user is free without restriction to define OPC data items and types of his choosing.

**Table 11: GUS Display Database Type Mappings**

DDB Item	Type	Description	NWDDDB Type
INT01 - INT256 / INT01G - INT256G	Integer	Storage for integer values in the local storage / Global storage	VT_I2, VT_I4
REAL01 - REAL256 / REAL01G - REAL256G	Float	Storage for floating point numbers in local storage / Global storage	VT_R4, VT_R8
BOOL01 - BOOL256 / BOOL01G - BOOL256G	Integer	Storage for Boolean values in the local storage / Global storage	VT_BOOL
STRING01 - STRING256 / STR01G - STR256G	String	Storage for string values in the local storage / Global storage	VT_BSTR
ENT01 - ENT256 / ENT01G - ENT256G	Point	Storage for a reference to a point in local storage / Global storage	VT_BSTR plus HCI Indirection
ENM01 - ENM256 / ENM01G - ENM256G	Enumeration	Storage for enumeration values in the local storage / Global storage. Only standard and custom enumerations can be stored here.	VT_BSTR

DDB Item	Type	Description	NWDDDB Type
VAR01 - VAR256 / VAR01G - VAR256G	Variable	Storage for a reference to a point.parameter in local / Global storage.	VT_BSTR plus HCI Indirection
DATIME1 - DATIME256 / DATIME1G - DATIME256G	Time	Storage for time values in the local storage / Global storage	VT_DATE
\$*, where * is a wildcard for any valid DDB actor.	Variable	Various	Not supported; use DDBs

### 15.9.3 Storage of LCN Internal IDs

The NWDDDB is capable of storing type 'Internal ID' of an LCN point for use in Changezone. This enables one display to communicate a point of interest for the second Changezone display. This approach is favored over storage of the 'External Name' of the LCN point because of the increased efficiency at the receiving end. That is, if the External Name of the point was stored instead, an interested user would first have to fetch the External Name from the database, and then convert to the Internal ID for use with the \$CZ\_ENTY actor. Instead, the Internal form is stored directly.

Also, in the example to follow, note the use of an entity DDB to extract the Internal ID. It has been verified that this cannot be done directly via a property at the point-level. Also, there is no known scripting Actor that provides this functionality. Both 'Internal' and 'External' are defined in GUS Scripting at the parameter level, but unless the parameter itself is of type Entity (that is, at the higher point-level), both of these properties have been shown to return the value for the parameter if applied to a parameter on the point. Furthermore, use of DDBs to extract the Internal ID and other properties relies on HOPCServer functionality, and the intent is to move to a pure HCI-only method for obtaining TPN Internal Ids.

Internal ID has been typed as four short integers, or 64 bits. It cannot be directly stored as a string and then converted back to its 64-bit representation. (Note that a Basic 'long' is 32 bits versus 64.) The method recommended is to both store and retrieve it as a string, letting Basic do the type coercion. The resultant string can then be stored to \$CZ\_ENTY.Internal. Note the use of \$CZ\_ENTY in this final step rather than a direct storage to a DDB entity type. \$CZ\_ENTY internally does a store to the DDB entity plus the additional step of initiating an LCN 'Update' for all the points associated with the Changezone.

### 15.9.4 Storage of LCN Type Entity and Type Variable

Type Entity, which represents an LCN entity or point name, and Type Variable, which represents an LCN parameter can both be stored to the NWDDDB Database as type VT\_BSTR.

Upon their retrieval, they can be then applied to the appropriate portion of a GUS HCI Client nameform through the use of 'HCI Bind Extensions' as discussed in a subsequent section of this document by that name to achieve indirection.

Note that the user can directly store both the 'Internal' and 'External' properties of type entity by using type string.



---

## 15.10 Support for Indirection

The NWDDB Server enables DDB Entity and Variable types to be stored as type VT\_BSTR.

When these components are subsequently retrieved from the NWDDB Database, indirection is facilitated through HCI Bind Extensions, which facilitates indirection at all levels of the GUS HCI Nameform. For more information, refer to the 'HCI Bind Extensions' section of this guide.

---

## 15.11 Human Interface

There are two aspects to NWDDDB from a human interface perspective: (a) the NWDDDB Database Input File definition and (b) the NWDDDB Server operations. A textual editor is used to achieve the definition of OPC data items. A new component, the NWDDDB Server is added to the GUS Display HCI Client Add-In Package to achieve the data access.

The NWDDDB Server is installed as a component under the GUS Display HCI Client Add-In Package.

At installation time, `nwddb_server.dss` is populated with test data. To add additional OPC data items, the user simply double-clicks on this file to open it under a textual editor such as Microsoft Notepad, and then makes his desired entries. Closing the file saves the new entries. Note that the definition of data items in the Input File is strictly an off-line capability. There are no provisions to add new items to the NWDDDB Database at run-time.

At package release time, the user is required to reconfigure the filename prefix '`nwddb_server`' to a name that correlates to his suite of associated displays. The '`.dss`' suffix must remain. This renamed NWDDDB Database Input File should then 'travel' with its corresponding suite of displays.

---

## 15.12 Programmatic Interfaces-NWDDDB Server

A new HCI/OPC Server for ES-T has been created called the NWDDDB Server. Its associated NWDDDB Database is read/written using standard OPC calls, and is accessible under the GUS HCI Client using nameform extensions. No new methods are added to communicate with the NWDDDB Server, except that the user is strongly encouraged to use 'nwddb' for the server alias in his Bind Method.

## 15.13 Scenarios and Examples - NWDDDB Server

### Related topics

- “Storage of LCN Internal IDs - Scenarios and Examples” on page 100
- “Storage of LCN Type Entity and Type Variable - Scenarios and Examples” on page 100
- “NWDDDB Server Startup and Shutdown” on page 102
- “NWDDDB Server Database Installation w/ Associated Package” on page 102
- “NWDDDB Database De-Installation w/ Associated Package” on page 102

### 15.13.1 Storage of LCN Internal IDs - Scenarios and Examples

Consider two displays, a Unit Display and a Changezone Display. Selecting a target on the Unit Display changes the point-of-interest in the Changezone Display.

```
Script on Target
Sub OnDisplayStartup
'Connect to NWDDDB Server
hci.nwddb.bind 'NWDDDB_Server'
End Sub
Sub OnLButtonUp()
'Set the entity to the target's point
'Note the use of an entity DDB to pull the Internal ID
'This cannot be done directly via a property on the point, or
'via a scripting Actor.
Dim szSaveInternalID As String
Set DispDB.Ent01 = lcn.ms_hist1
szSaveInternalID = DispDB.Ent01.Internal
'Save the Internal ID of the point-of-interest to 'InternalID', which is type VT_BSTR
hci.nwddb.ChangeZoneSuite.InternalID = szSaveInternalID
End Sub
Script on Changezone Display
Sub OnDisplayStartup
'Connect to NWDDDB Server
hci.nwddb.bind 'NWDDDB_Server'
End Sub
Sub OnDataChange()
'ODC Tripped; there is a new point of interest
'Get the InternalID from the NWDDDB Database
Dim szGetString As String
szGetString = hci.nwddb.ChangeZoneSuite.InternalID
DispDB.[{sz_enty}.Internal = szGetString
End Sub
```

### 15.13.2 Storage of LCN Type Entity and Type Variable - Scenarios and Examples

The following scripting examples illustrate the use of NWDDDB Server in conjunction with HCI Bind Extensions to store and use LCN Type Entity and Type Variable. This same principle could be applied to LCN nameforms as shown in the third example, with the restraint of having to use a DDB to facilitate the indirection.

Example 1- Entity Type applied to HCI Nameform with Deferred Server Binding - Preferred Approach

Display #1

```
Sub OnDisplayStartup
'Bind to servers
hci.nwddb.bind 'NWDDDB_Server'
Msgbox 'Bound to GUS NWDDDB Server'
hci.s2.bind 'Hci_TPNServer'
Msgbox 'Bound to TPN Server'
End Sub
Sub OnLButtonClick
'Save Entity
Dim sEntity As String
sEntity = 'ramp0001'
'Store to NWDDDB Database
hci.nwddb.MyPackage.MyEntity = sEntity
End Sub
```

## Display #2

```

Sub OnDisplayStartup
'Bind to servers
hci.nwddb.bind 'NWDDDB_Server'
Msgbox 'Bound to GUS NWDDDB Server'
Msgbox 'Bind to TPN Server purposely deferred'
End Sub
Sub OnLButtonClick
'Retrieve Entity
Dim sEntity As String
sEntity = hci.nwddb.MyPackage.MyEntity
'Unbind server if already bound;
'However, we are not yet bound - preferred approach
'Bind entity to indirect nameform
hci.s2.pointname.bind sEntity
'Bind server
hci.s2.bind 'Hci_TPNServer'
'Read data
me.text = hci.s2.pointname.sp
End Sub

```

## Example 2. Variable Type applied to HCI Nameform

## Display #1

```

Sub OnDisplayStartup
'Bind to servers
hci.nwddb.bind 'NWDDDB_Server'
Msgbox 'Bound to GUS NWDDDB Server'
hci.s2.bind 'Hci_TPNServer'
Msgbox 'Bound to TPN Server'
End Sub
Sub OnLButtonClick
'Save Variable
Dim sVariable As String
sVariable = 'ramp0001.sp'
'Store to NWDDDB Database
hci.nwddb.MyPackage.MyVariable = sVariable
End Sub

```

## Display #2

```

Sub OnDisplayStartup
'Bind to servers
hci.nwddb.bind 'NWDDDB_Server'
Msgbox 'Bound to GUS NWDDDB Server'
hci.s2.bind 'Hci_TPNServer'
Msgbox 'Bound to TPN Server'
End Sub
Sub OnLButtonClick
'Retrieve Variable
Dim sVariable As String
sVariable = hci.nwddb.MyPackage.MyVariable
'Unbind server
hci.s2.bind ''
'Bind entity to indirect nameform
hci.s2.pointparam.bind sVariable
'Rebind server
hci.s2.bind 'Hci_TPNServer'
'Read data
me.text = hci.s2.pointparam
End Sub

```

## Example 3. Variable Type applied to LCN Nameform

## Display #1

```

Sub OnDisplayStartup
'Bind to server
hci.nwddb.bind 'NWDDDB_Server'
Msgbox 'Bound to GUS NWDDDB Server'
End Sub
Sub OnLButtonClick
'Save Variable
Dim sVariable As String
sVariable = 'ramp0001.sp'
'Store to NWDDDB Database

```

```
hci.nwddb.MyPackage.MyVariable = sVariable
End Sub
```

#### Display #2

```
Sub OnDisplayStartup
'Bind to server
hci.nwddb.bind 'NWDDDB_Server'
Msgbox 'Bound to GUS NWDDDB Server'
End Sub
Sub OnButtonClick
'Retrieve Variable
Dim sVariable As String
sVariable = hci.nwddb.MyPackage.MyVariable
'Read data
set dispdb.var01 = getvar(sVariable)
me.text = dispdb.var01
End Sub
```

### 15.13.3 NWDDDB Server Startup and Shutdown

NT Power up of node occurs.

User executes GUS display with Bind request to NWDDDB Server.

COM notes that there is no running instance of NWDDDB Server, so that CoCreateInstance call results in initial creation of the server. (Subsequent Bind calls will simply return an interface pointer to this single running instance.)

As part of its initialization, NWDDDB Server will call Lock Server so that it will NOT be deleted by COM when there are no bound clients.

Server runs under the NWDDDBServer dedicated account, so that it can remain running across user logins/logouts.

Node is shut down

NWDDDB Server shuts down

### 15.13.4 NWDDDB Server Database Installation w/ Associated Package

For more information, refer to the 'NWDDDB Database and Package Administration' section of this guide.

TrendGrids Package install scripts copy NWDDDB Database 'TrendGrids.dss' to C:\ProgramData\Honeywell\Checkpoints\OCX.

TrendGrids Package install scripts add the following entry to the NWDDDB Server's log file C:\ProgramData\Honeywell\Checkpoints\TrendGrids.dss'.

At startup time, NWDDDB Server creates concatenated .dss file as follows:

Open each element in the log file, C:\ProgramData\Honeywell\Checkpoints\nwddb\_server.log

Open \ C:\ProgramData\Honeywell\Checkpoints\nwddb\_server.dss

Open & append all elements in log file; for this example, only C:\ProgramData\Honeywell\Checkpoints\TrendGrids.dss.

Populate cache with contents of new concatenated .dss file.

### 15.13.5 NWDDDB Database De-Installation w/ Associated Package

For more information, refer to the 'NWDDDB Database and Package Administration' section of this guide.

TrendGrids Package de-install scripts remove NWDDDB Database 'TrendGrids.dss' from C:\ProgramData\Honeywell\Checkpoints\OCX.

TrendGrids Package install scripts find/delete the following entry from the NWDDB Server's log file: C:\ProgramData\Honeywell\Checkpoints \TrendGrids.dss'.





# 16 HCI Bind Extensions

The GUS HCI Nameform has been augmented to support a simple, powerful form of indirection that exploits the late binding capabilities of OPC. Through HCI Bind Extensions, the GUS HCI Client together with GUS Scripting enables indirection at all levels of the HCI/OPC data item:

1. Server Namestring
2. Point Name
3. Parameter Name

The semantics of the GUS HCI Client's 'Bind' Method has been extended to allow 'redirection' of any portion of an HCI nameform. Previously, the Bind Method was supported only on the server name portion of the GUS HCI Nameform. The semantics of 'Bind' on other parts of the GUS HCI Nameform now have very similar semantics to the 'Bind' on the server name portion.

Under HCI Bind Extensions, the Bind Method allows the user to associate a symbolic name with an actual name for any segment of the nameform. In other words, the semantics of 'Bind' is now 'define the actual name to be used for this alias'. A major side benefit of HCI Bind Extensions is that it allows users to reduce or eliminate the need to use the bracket syntax as a mechanism to reference names that are not lexically valid in BasicScript. The user simply indicates lexically valid symbolic names for those points that can not be referenced without the bracket syntax. Then, the user binds the symbolic names with the actual names prior to binding the server.

## 16.1 Syntax

The following is the syntax and semantics of the GUS HCI Client's Extended 'Bind' Method, which indicates that it can be applied at any level of the GUS HCI Nameform:

```
hci.<serveralias>{.<namealias>}.{<namealias>}.bind <bindstring>
```

where:

<serveralias> : a symbolic name/alias representing an HCI/OPC Server

<namealias> : a symbolic name/alias for subordinates on the nameform. This can include a point name, a parameter name, or a combination of the above.

<bindstring> : a string containing the actual name to use when the alias is bound. This can be any expression that evaluates to a string.

Notes

1. No changes are made to the syntax of the GUS HCI Nameform.
2. The default binding for a symbol is the name of the symbol. That is, if no Bind Method occurs on a symbol prior to its server being bound, the name will be treated as a direct reference.
3. When binding a name, its server may be in a bound or an unbound state. That is, you can change the binding of a symbol independently of the bind state of the server. If the server is currently in bound state, collection groups are updated without rebinding the server..
4. The Bind Method applies only to the name segment on which the bind occurs.
5. The Bind Method can be used at any level in a nameform.
6. Names can be unbound by assigning an empty string. If an unbound name segment results in the binding of an invalid name, the nameform will have a status of 'Configuration Error'.
7. Binding to a non-existent point will produce bad status on returned data. The data change event gives the display a chance to report the bad status.



### Attention

For Performance reasons, display authors who will use HCI Bind Extensions are strongly encouraged to place their Bind statements in OnDisplayStartup scripts, and only bind to their servers once Bind Methods have first been completed for nameform subordinates. Another reason for resolving at display startup is to avoid configuration errors for all unbound aliases.

---

## 16.2 'Binding' Property

The property 'Binding' returns the current binding of an alias. Changes in the 'Binding' property do not trigger execution of the onDataChange scripts (including the VTO values).

---

# 16.3 Programmatic Interfaces - Binding Extensions

Two upgrades have been made to GUS HCI Client's Programmatic Interfaces to support extended binding: (a) the Bind method has been augmented and (b) a new property called 'Binding' has been added.

*Example Bind Method calls*

Bind server alias to HCI/OPC Server

---

hci.s1.bind 'template\_server1'

---

Bind point alias to point:

---

hci.s1.point.bind 'p1'

---

Bind pointparam alias to point/parameter:

---

hci.s1.pointparam.bind 'p1.pv'


---

Bind param alias to parameter:

---

hci.s1.p1.param.bind 'pv'

---

 **Attention**  
Note that the parameter alias must be defined relative to a given point, or point alias.

---

---

The 'Binding' Property has been added to return the current alias. This property can be applied to any portion of the GUS HCI Nameform except for property. For an unbound name segment, this property returns the name of the name segment. For a bound name segment, this property returns the alias for the bound segment. This is illustrated in the samples below.

---

## 16.4 Scenarios and Examples - Binding Extensions

### Related topics

“Example 1: Changing the binding” on page 109

“Example 2: Binding before and after the 'Bind' statement” on page 110

### 16.4.1 Example 1: Changing the binding

The following depicts two text objects, one that displays the 'sp' parameter, and a second that displays the 'pv' parameter. The display will start up initially showing the values of a100.sp and a100.pv. When the user clicks on a dedicated object to update points, he is prompted for a new point name. After entering a point name, the text objects will begin showing the value of the sp and pv parameter for the new point. This example shows only two parameters being referenced. A typical faceplate display may reference 20 parameters. In that case, the switch between points would still only require minimal code to bind the new point name to the alias. Also consider a parameterized display. It would be very simple to obtain a point name from a well know registry entry, then bind a GUS HCI Nameform to that point name.

```
Dim szServername as String
Sub OnDisplayStartup()
szServername = 'bigserver'
hci.server1.boiler.bind 'a100'
hci.server1.bind szServername
End Sub
```

#### ‘Text Object 1

```
Sub OnDataChange()
On Error GoTo Error_Handler
me.text = hci.server1.boiler.sp
Exit Sub
Error_Handler:
If (Err.Number = DAS_COMMUNICATION_ERROR) Then
‘User code to handle communication errors’
Else
‘User code to handle other errors
‘...
End If
End Sub
```

#### ‘Text Object 2

```
Sub OnDataChange()
On Error GoTo Error_Handler
me.text = hci.server1.boiler.pv
Exit Sub
Error_Handler:
If (Err.Number = DAS_COMMUNICATION_ERROR) Then
‘User code to handle communication error’
Else
‘User code to handle other errors
‘...
End If
End Sub
```

#### ‘Change point Text Object

```
Sub OnLButtonClick()
Dim szVariable as String
szVariable = inputbox$('Input Temperature point')
hci.server1.boiler.bind szVariable
End Sub
```

### 16.4.2 Example 2: Binding before and after the 'Bind' statement

The following script shows the server binding both before and after the 'Bind' statement. The user can see the update by a second click of the mouse button.

```
Sub OnButtonClick
Me.text = hci.s1.bind 'Displays current server bound to s1, or '' if server is unbound
Hci.s1.bind 'template_server1'
Msgbox 'Bound to template_server1'
Me.text = hci.s1.bind 'Displays template_server1
End Sub
```

#### Example 3

Example Binding Property usages follow.

Property applied to unbound server component:

Me.text = hci.s1.binding 'returns ''

Property applied to server component bound to 'template\_server1':

Me.text = hci.s1.binding 'returns 'template\_server1'

Property applied to point name / parameter name combination prior to binding:

Me.text = hci.s1.pointparam.binding 'returns 'pointparam'

Property applied to point name / parameter name combination after binding to 'p1.pv'

Me.text = hci.s1.pointparam.binding 'returns 'p1.pv'

Property applied to point name prior to binding:

Me.text = hci.s1.pointname.binding 'returns 'pointname'

Property applied to point name after binding to 'p1'

Me.text = hci.s1.pointname.binding 'returns 'p1'

Property applied to parameter name prior to binding

Me.text = hci.s1.p1.paramname.binding 'returns 'paramname'

Property applied to parameter name after binding to 'pv'

Me.text = hci.s1.p1.paramname.binding 'returns 'pv'

---

## 16.5 Performance Requirements

Binding to subordinates of the GUS HCI Nameform (other than the server component) involve only the internal GUS Scripting symbol table if the server is unbound. If the server is bound then all existing OPC collection groups that are affected by the change need to be updated.





# 17 Appendices

## **Related topics**

“Appendix A: Comparison of HCI and HOPC data access” on page 114

“Appendix B: Default NWDDDB Database” on page 117

## 17.1 Appendix A: Comparison of HCI and HOPC data access

### 17.1.1 HOPC & HCI/OPC Data Access Capabilities

The functions and data collections supported for LCN data access via HOPC were used as a premise for the support that would be provided for communication with HCI/OPC-based servers in GUS Display Builder. There are some differences, which are detailed in the table below.

GUS Named Data Access provides 'Property' extensions for both HOPC and HCI/OPC data access. For a detailed description pertaining to HOPC Data Server offerings, reference 'Properties and methods on data objects/Properties of a Parameter Object' under the GUS Display Builder On-line Help.

**Table 12: HOPC & HCI/OPC Data Access Capabilities**

Capability	Supported in HOPC Data Access	Supported in HCI/OPC data Access
'Binding' Property		X
'External' Property	X	
'ID' Property	X	
'Internal' Property	X	
'Name' Property	X	
'NodeType' Property	X	
'Quality' Property		X
'Status' Property	X	X
'TimeStamp' Property		X
'Type' Property	X	X
'Value' Property	X	X
'ValueNoError' Property		X
Constant Indexing	X	X
Demand read of all groups via 'Update' method with input parameter of zero	X	X
Demand read of multi-item groups via 'Update' method with input parameter specifying desired group	X	X
Demand reads from device	X	X
Demand writes to device	X	X
Dynamic indexing	X	X
Dynamic run-time connectivity to servers via 'Bind' method		X
Entity ID parameter reads/writes	X	
Error detection and error handling	X	X
Fast update group	X	X
Indication of server failure on GUS display	X	X
Indirection	X	X
Keylock propagation to server	X	X
Multiple client support	X	X
Multiple server connectivity		X

Capability	Supported in HOPC Data Access	Supported in HCI/OPC data Access
OnDataChange cached reads	X	X
OnPeriodicUpdate cached reads	X	X
Validation of point/parameters at the data server during display validation	X	

### 17.1.2 Consistent Error Handling Across Servers

The GUS HCI Client makes every effort to ensure consistency in the error status displayed to the user across both HOPC and HCI/OPC servers by mapping like errors to the same common scripting error status. For more information, refer to “Table 4: GUS HCI Client Errors and GUS Display Builder Scripting Status”.

### 17.1.3 Mapping from LCN Data Types to BasicScript Data Types

The following information is based on the GUS Display Builder's on-line HELP to demonstrate the mapping of LCN Data Types to their BasicScript Data Type for scripting under the GUS Display Builder. The table has been extended to demonstrate the eventual mapping of LCN Data Type to type VARIANT.

Care should be taken when manipulating LCN data within scripts. LCN data types must in some cases be mapped into the appropriate Basic data types. This table describes in general how these types would map. Note that under BasicScript, some additional types are offered.

Table 13: Mapping Data Types

LCN Data Type	BasicScript Data Type	VARIANT Data Type
Boolean	Boolean	VT_BOOL
Not Supported	Currency	VT_CY
Date/Time	Date	VT_DATE
Real	Double	VT_R8
Integer	Integer	VT_I2
Not Supported	Long	VT_I4
Not Supported	Object	VT_DISPATCH
Real	Single	VT_R4
String	String	VT_BSTR
Not Supported	Variant	VT_VARIANT
Enumeration	String	VT_BSTR
SD Enumeration	String	VT_BSTR
Unknown	String	VT_BSTR

### 17.1.4 Display Validation Differences

In the case of HOPC data references, there is an extra step to the validation process, which involves verifying that the data items exist on the server (TPN). For HCI/OPC data items, this verification is postponed until run-time, which is when bindings are made to server, and data items can be validly referenced.

### **17.1.5 Demand Groups**

HCI and HOPC data items can be mixed in the same Demand Group, but two different scripts statements are needed to refresh the data items, one for HCI (hci.Update <DemandGroupID>) and one for HOPC (lcn.Update <DemandGroupID>).

### **17.1.6 Fast Group Update**

HOPC data server implements 'Fast Update' by internally switching group collection rate down to 1 second.

### **17.1.7 Indirect references**

HOPC indirect data references can be stored in DispDB or at the parameter of the TPN point.

HCI point/parameter indirection can be achieved using the 'Bind' extensions.

## 17.2 Appendix B: Default NWDDDB Database

```
// -----
// DataItem Var.Type Qual. Value
// -----
UI1 17 0xc0 f
I1 16 0xc0 2
I2 2 0xd8 222
I4 3 0xc0 444
I4.value 3 0xc0 444
R4 4 0xc0 444.44
R8 5 0xc0 888.88
//CY Value is in (units, fractional_units), where fractional units = desired result * 10000
//Thus, this is the representation for $4321.78
CY 6 0xc0 4321 78
CY.strval 8 0xc0 '$4321.78'
// DATE is typedef double DATE
CurrentTime 7 0xc0 '6:33:20 PM 6/19/97'
Time11:11 7 0xc0 '11:11:11 AM 6/19/97'
BSTR 8 0xc0 'BSTR.value'
ERROR 10 0xc0 0x8007000E
ERROR.strval 8 0xc0 '0x8007000E == 2147942414'
ERROR.string 8 0xc0 'Ran out of memory'
BOOL.false 11 0xc0 0
BOOL.true 11 0xc0 -1
ENUM.color 2 0xc0 0
R4ARRAY 8192 0xc0 5 0 12 4 201.1 4 202.2 4 203.3 4 204.4 4 205.5
ARRAY 8192 0xc0 3 1 12 2 987 4 876.6 8 'array string'
Bad 4 0x0c 0.0
Uncertain 5 0x58 432.1
// Arrays of VARIANTS
// (# of elements, lowerbound, type for all (variant), type for data, data, type for data, data,
and so on.)
//
ARRAY_of_R8 8192 0xc0 4 0 12 5 303.1 5 304.2 5 305.3 5 306.4
ARRAY_of_I2 8192 0xc0 10 0 12 2 2 2 3 2 5 2 7 2 11 2 13 2 17 2 19 2 23 2 29
ARRAY_of_I1 8192 0xc0 5 0 12 16 1 16 2 16 3 16 4 16 5
//
// These arrays contain elements of the exact type, not VARIANTS
// (# of elements, lowerbound, type for all, data)
I1_ARRAY 8192 0xc0 10 0 16 1 2 3 4 5 6 7 8 9 10
I2_ARRAY 8192 0xc0 10 0 2 1 2 3 4 5 6 7 8 9 10
R8_ARRAY 8192 0xc0 10 0 5 .1 .2 .3 .4 .5 .6 .7 .8 .9 1.0
BSTR_ARRAY 8192 0xc0 10 0 8 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k'
GUS.CWDDDB_Item 8 0xc0 'Console-wide DDB Data Item'
```



# 18 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>.

---

## 18.1 Documentation feedback

You can find the most up-to-date documents on the Honeywell Process Solutions support website at:

<http://www.honeywellprocess.com/support>

If you have comments about Honeywell Process Solutions documentation, send your feedback to:

[hpsdocs@honeywell.com](mailto:hpsdocs@honeywell.com)

Use this email address to provide feedback, or to report errors and omissions in the documentation. For immediate help with a technical problem, contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.



---

## 18.2 How to report a security vulnerability

For the purpose of submission, a security vulnerability is defined as a software defect or weakness that can be exploited to reduce the operational or security capabilities of the software.

Honeywell investigates all reports of security vulnerabilities affecting Honeywell products and services.

To report a potential security vulnerability against any Honeywell product, please follow the instructions at:

<https://honeywell.com/pages/vulnerabilityreporting.aspx>

Submit the requested information to Honeywell using one of the following methods:

- Send an email to [security@honeywell.com](mailto:security@honeywell.com).
- or
- Contact your local Honeywell Process Solutions Customer Contact Center (CCC) or Honeywell Technical Assistance Center (TAC) listed in the “Support and other contacts” section of this document.

---

## 18.3 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>.

---

## 18.4 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>.

