Documentation Library

Cogent C API and Utilities

Version 7.2

Cogent Real-Time Systems, Inc.

September 16, 2010

Cogent C API and Utilities: Version 7.2

The Application Program Interface and several utilities used by the Cogent tools.

Published September 16, 2010
Cogent Real-Time Systems, Inc.

162 Guelph Street, Suite 253
Georgetown, Ontario
Canada, L7G 5X7

Toll Free: 1 (888) 628-2028
Tel: 1 (905) 702-7851
Fax: 1 (905) 702-7850

Information Email: info@cogent.ca
Tech Support Email: support@cogent.ca
Web Site: www.cogent.ca

Revision History

Revision 7.2-1 September 2007
Updated to maintain compatibility with Windows DataHubs.
Revision 6.2-1 February 2005
Removed synchronous TCP functions.
Revision 5.0-1 August 2004
Compatible with Cascade DataHub and Cascade Connect Version 5.0.
Revision 4.1-1 September 2002
Added Cascade Historian reference entries.
Revision 4.0-2 October 2001
Added function reference entries.
Revision 4.0-1 September 2001
Source code compatible across QNX 4, QNX 6, and Linux.
Revision 3.0-1 September 2000
Initial release of documentation.

Copyright, trademark, and software license information.

Copyright Notice

Trademark Notice

END-USER LICENSE AGREEMENT FOR COGENT SOFTWARE

IMPORTANT - READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Cogent Real-Time Systems Inc. ("Cogent") of 162 Guelph Street, Suite 253, Georgetown, Ontario, L7G 5X7, Canada (Tel: 905-702-7851, Fax: 905-702-7850), from whom you acquired the Cogent software product(s) ("SOFTWARE PRODUCT" or "SOFTWARE"), either directly from Cogent or through one of Cogent's authorized resellers.
The SOFTWARE PRODUCT includes computer software, any associated media, any printed materials, and any "online" or electronic documentation. By installing, copying or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree with the terms of this EULA, Cogent is unwilling to license the SOFTWARE PRODUCT to you. In such event, you may not use or copy the SOFTWARE PRODUCT, and you should promptly contact Cogent for instructions on return of the unused product(s) for a refund.

SOFTWARE PRODUCT LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1. **EVALUATION USE:** This software is distributed as "Free for Evaluation", and with a per-use royalty for Commercial Use, where "Free for Evaluation" means to evaluate Cogent's software and to do exploratory development and "proof of concept" prototyping of software applications, and where "Free for Evaluation" specifically excludes without limitation:
    i. use of the SOFTWARE PRODUCT in a business setting or in support of a business activity,
    ii. development of a system to be used for commercial gain, whether to be sold or to be used within a company, partnership, organization or entity that transacts commercial business,
    iii. the use of the SOFTWARE PRODUCT in a commercial business for any reason other than exploratory development and "proof of concept" prototyping, even if the SOFTWARE PRODUCT is not incorporated into an application or product to be sold,
    iv. the use of the SOFTWARE PRODUCT to enable the use of another application that was developed with the SOFTWARE PRODUCT,
    v. inclusion of the SOFTWARE PRODUCT in a collection of software, whether that collection is sold, given away, or made part of a larger collection.
    vi. inclusion of the SOFTWARE PRODUCT in another product, whether or not that other product is sold, given away, or made part of a larger product.

2. **COMMERCIAL USE:** COMMERCIAL USE is any use that is not specifically defined in this license as EVALUATION USE.

3. **GRANT OF LICENSE:** This EULA covers both COMMERCIAL and EVALUATION USE of the SOFTWARE PRODUCT. Either clause (A) or (B) of this section will apply to you, depending on your actual use of the SOFTWARE PRODUCT. If you have not purchased a license of the SOFTWARE PRODUCT from Cogent or one of Cogent's authorized resellers, then you may not use the product for COMMERCIAL USE.
    A. **GRANT OF LICENSE (EVALUATION USE):** This EULA grants you the following non-exclusive rights when used for EVALUATION purposes:

Software: You may use the SOFTWARE PRODUCT on any number of computers, either stand-alone, or on a network, so long as every use of the SOFTWARE PRODUCT is for EVALUATION USE. You may reproduce the SOFTWARE PRODUCT, but only as reasonably required to install and use it in accordance with this LICENSE or to follow your normal back-up practices.

Subject to the license expressly granted above, you obtain no right, title or interest in or to the SOFTWARE PRODUCT or related documentation, including but not limited to any copyright, patent, trade secret or other proprietary rights therein. All whole or partial copies of the SOFTWARE PRODUCT remain property of Cogent and will be considered part of the SOFTWARE PRODUCT for the purpose of this EULA.

Unless expressly permitted under this EULA or otherwise by Cogent, you will not:

i. use, reproduce, modify, adapt, translate or otherwise transmit the SOFTWARE PRODUCT or related components, in whole or in part;

ii. rent, lease, license, transfer or otherwise provide access to the SOFTWARE PRODUCT or related components;

iii. alter, remove or cover proprietary notices in or on the SOFTWARE PRODUCT, related documentation or storage media;

iv. export the SOFTWARE PRODUCT from the country in which it was provided to you by Cogent or its authorized reseller;

v. use a multi-processor version of the SOFTWARE PRODUCT in a network larger than that for which you have paid the corresponding multi-processor fees;

vi. decompile, disassemble or otherwise attempt or assist others to reverse engineer the SOFTWARE PRODUCT;

vii. circumvent, disable or otherwise render ineffective any demonstration time-outs, locks on functionality or any other restrictions on use in the SOFTWARE PRODUCT;

viii. circumvent, disable or otherwise render ineffective any license verification mechanisms used by the SOFTWARE PRODUCT;

ix. use the SOFTWARE PRODUCT in any application that is intended to create or could, in the event of malfunction or failure, cause personal injury or property damage; or

x. make use of the SOFTWARE PRODUCT for commercial gain, whether directly, indirectly or incidentally.

B. **GRANT OF LICENSE (COMMERCIAL USE):** This EULA grants you the following non-exclusive rights when used for COMMERCIAL purposes:

Software: You may use the SOFTWARE PRODUCT on one computer, or if the SOFTWARE PRODUCT is a multi-processor version - on one node of a network, either: (i) as a development systems for the purpose of creating value-added software applications in accordance with related Cogent documentation; or (ii) as a single run-time copy for use as an integral part of such an application. This includes reproduction and configuration of the SOFTWARE PRODUCT, but only as reasonably required to install and use it in association with your licensed processor or to follow your normal back-up practices.

Storage/Network Use: You may also store or install a copy of the SOFTWARE PRODUCT on one computer to allow your other computers to use the SOFTWARE PRODUCT over an internal network, and distribute the SOFTWARE PRODUCT to your other computers over an internal network. However, you must acquire and dedicate a license for the SOFTWARE PRODUCT for each computer on which the SOFTWARE PRODUCT is used or to which it is distributed. A license for the SOFTWARE PRODUCT may not be shared or used concurrently on different computers.

Subject to the license expressly granted above, you obtain no right, title or interest in or to the SOFTWARE PRODUCT or related documentation, including but not limited to any copyright, patent, trade secret or other proprietary rights therein. All whole or partial copies of the SOFTWARE PRODUCT remain property of Cogent and will be considered part of the SOFTWARE PRODUCT for the purpose of this EULA.

Unless expressly permitted under this EULA or otherwise by Cogent, you will not:

i. use, reproduce, modify, adapt, translate or otherwise transmit the SOFTWARE PRODUCT or related components, in whole or in part;

ii. rent, lease, license, transfer or otherwise provide access to the SOFTWARE PRODUCT or related components;

iii. alter, remove or cover proprietary notices in or on the SOFTWARE PRODUCT, related documentation or storage media;

iv. export the SOFTWARE PRODUCT from the country in which it was provided to you by Cogent or its authorized reseller;

v. use a multi-processor version of the SOFTWARE PRODUCT in a network larger than that for which you have paid the corresponding multi-processor fees;

vi. decompile, disassemble or otherwise attempt or assist others to reverse engineer the SOFTWARE PRODUCT;

vii. circumvent, disable or otherwise render ineffective any demonstration time-outs, locks on functionality or any other restrictions on use in the SOFTWARE PRODUCT;

viii. circumvent, disable or otherwise render ineffective any license verification mechanisms used by the SOFTWARE PRODUCT, or

ix. use the SOFTWARE PRODUCT in any application that is intended to create or could, in the event of malfunction or failure, cause personal injury or property damage.

4. **WARRANTY:** Cogent cannot warrant that the SOFTWARE PRODUCT will function in accordance with related documentation in every combination of hardware platform, software environment and SOFTWARE PRODUCT configuration. You acknowledge that software bugs are likely to be identified when the SOFTWARE PRODUCT is used in your particular application. You therefore accept the responsibility of satisfying yourself that the SOFTWARE PRODUCT is suitable for your intended use. This includes conducting exhaustive testing of your application prior to its initial release and prior to the release of any related hardware or software modifications or enhancements.

   Subject to documentation errors, Cogent warrants to you for a period of ninety (90) days from acceptance of this EULA (as provided above) that the SOFTWARE PRODUCT as delivered by Cogent is capable of performing the functions described in related Cogent user documentation when used on appropriate hardware. Cogent also warrants that any enclosed disk(s) will be free from defects in material and workmanship under normal use for a period of ninety (90) days from acceptance of this EULA. Cogent is not responsible for disk defects that result from accident or abuse. Your sole remedy for any breach of warranty will be either: i) terminate this EULA and receive a refund of any amount paid to Cogent for the SOFTWARE PRODUCT, or ii) to receive a replacement disk.

5. **LIMITATIONS:** Except as expressly warranted above, the SOFTWARE PRODUCT, any related documentation and disks are provided "as is" without other warranties or conditions of any kind, including but not limited to implied warranties of merchantability, fitness for a particular purpose and non-infringement. You assume the entire risk as to the results and performance of the SOFTWARE PRODUCT. Nothing stated in this EULA will imply that the operation of the SOFTWARE PRODUCT will be uninterrupted or error free or that any errors will be corrected. Other written or oral statements by Cogent, its representatives or others do not constitute warranties or conditions of Cogent.

   In no event will Cogent (or its officers, employees, suppliers, distributors, or licensors: collectively "Its Representatives") be liable to you for any indirect, incidental, special or consequential damages whatsoever, including but not limited to loss of revenue, lost or damaged data or other commercial or economic loss, arising out of any breach of this EULA, any use or inability to use the SOFTWARE PRODUCT or any claim made by a third party, even if Cogent (or Its Representatives) have been advised of the possibility of such damage or claim. In no event will the aggregate liability of Cogent (or that of Its Representatives) for any damages or claim, whether in contract, tort or otherwise, exceed the amount paid by you for the SOFTWARE PRODUCT.

   These limitations shall apply whether or not the alleged breach or default is a breach of a fundamental condition or term, or a fundamental breach. Some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, or certain limitations of implied warranties. Therefore the above limitation may not apply to you.

6. **DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS:**

   Separation of Components. The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.

Termination. Without prejudice to any other rights, Cogent may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such an event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

7. **UPGRADES:** If the SOFTWARE PRODUCT is an upgrade from another product, whether from Cogent or another supplier, you may use or transfer the SOFTWARE PRODUCT only in conjunction with that upgrade product, unless you destroy the upgraded product. If the SOFTWARE PRODUCT is an upgrade of a Cogent product, you now may use that upgraded product only in accordance with this EULA. If the SOFTWARE PRODUCT is an upgrade of a component of a package of software programs which you licensed as a single product, the SOFTWARE PRODUCT may be used and transferred only as part of that single product package and may not be separated for use on more than one computer.
8. **COPYRIGHT:** All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text and ’applets", incorporated into the SOFTWARE PRODUCT), any accompanying printed material, and any copies of the SOFTWARE PRODUCT, are owned by Cogent or its suppliers. You may not copy the printed materials accompanying the SOFTWARE PRODUCT. All rights not specifically granted under this EULA are reserved by Cogent.
9. **PRODUCT SUPPORT:** Cogent has no obligation under this EULA to provide maintenance, support or training.
10. **RESTRICTED RIGHTS:** Use, duplication, or disclosure by the U.S. Government is subject to restrictions as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (OCT 1988), FAR 12.212(a)(1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as appropriate. Manufacturer is Cogent Real-Time Systems Inc. 162 Guelph Street, Suite 253, Georgetown, Ontario, L7G 5X7, Canada.
11. **GOVERNING LAW:** This Software License Agreement is governed by the laws of the Province of Ontario, Canada. You irrevocably attorn to the jurisdiction of the courts of the Province of Ontario and agree to commence any litigation that may arise hereunder in the courts located in the Judicial District of Peel, Province of Ontario.

Table of Contents

I. Programmers Manual xiii

1. Introduction 1
   - 1.1. When to use the different Cogent APIs 1
   - 1.2. Function Naming Conventions 2
   - 1.3. System Requirements 2
   - 1.4. Download and Installation 2
     - 1.4.1. QNX 4 ..... **??**
     - 1.4.2. QNX 6 ..... **??**
     - 1.4.3. Linux ..... **??**
     - 1.4.4. Installed file locations ..... **??**
     - 1.4.5. Installing licenses ..... **??**
   - 1.5. Cogent Product Integration 4
   - 1.6. Where can I get help? 4
2. Point Structure, Storage, and Manipulation 6
   - 2.1. Creating Points 6
   - 2.2. Maintaining a Point Hash Table 6
   - 2.3. Accessing and Copying Point Values 7
   - 2.4. Memory Allocation and String Values 7
3. Interprocess Communication 8
   - 3.1. Connections and Channels 8
   - 3.2. Task Structure Caching 8
   - 3.3. Messages 8
   - 3.4. Cascade NameServer Functions 9
   - 3.5. Photon Functions 10
   - 3.6. Pulses and Timers 10
   - 3.7. Cascade QueueServer Functions 10
   - 3.8. Receiving Messages and Events 11
   - 3.9. Replying to Messages 11
   - 3.10. Sending Messages 11
   - 3.11. Task Structures 12
   - 3.12. Working with TCP/IP 12
4. The Cascade NameServer 13
   - 4.1. Domains 13
   - 4.2. Locating Other Tasks on the Network 13
   - 4.3. Task Started and Stopped Messages 14
5. Communicating with the Cascade DataHub 15
   - 5.1. Exceptions 15
   - 5.2. Echoes 15
   - 5.3. Non-Existent Cascade DataHub Points 15
   - 5.4. Parsing Point Messages 15
   - 5.5. Optimizing Throughput 16
   - 5.6. Point Size Limit 16
   - 5.7. Cascade DataHub API Code Examples 17
     - 5.7.1. Reading from the Cascade DataHub 17
     - 5.7.2. Writing data to the Cascade DataHub 20
     - 5.7.3. Registering for exceptions from the Cascade DataHub 24
     - 5.7.4. A sample makefile definition 29
6. The Cascade Historian 30
   - 6.1. Command/Function Correspondence 30
   - 6.2. Binary Data Buffer Functions 31

7. Cogent Driver Specifications 32
7.1. Cogent Driver Functions 32
7.2. Hilscher Fieldbus CIF Card 32
7.2.1. I/O Block Functions 32
7.2.2. Control Block Functions 32
7.2.3. Status Block Functions 33
A. GNU General Public License 35
B. GNU Lesser General Public License 41
II. Reference 49
I. Utilities 50
lsend, **gsend** 51
nsnames 53
nserve 55
qserve 57
II. Data Types 58
`HI_stVALUE` 59
`PT_stCPOINT` 60
`PT_TYPE`, `PT_uVALUE` 61
`ST_STATUS` 62
III. Cascade DataHub Functions 63
`DH_AppendString` 64
`DH_CreatePoint` 65
`DH_FindPointAddress` 66
`DH_FormatPoint` 67
`DH_ParsePointMsg` 69
`DH_ParsePointString` 71
`DH_PointAdd`, `DH_PointDivide`, `DH_PointMultiply` 73
`DH_ReadPoint`, `DH_ReadExistingPoint` 75
`DH_RegisterAllPoints` 77
`DH_RegisterPoint`, `DH_RegisterExistingPoint` 78
`DH_SendPointMessage` 80
`DH_SetLock`, `DH_SetSecurity` 81
`DH_SetReceiveFormat` 82
`DH_SetTransmitFormat` 84
`DH_UnregisterPoint` 85
`DH_WritePoint`, `DH_WriteExistingPoint`, `DH_WriteExistingPoints` 86
IV. Cogent Driver Functions 88
`DR_ApCloseIPC` 89
`DR_ApCommand` 90
`DR_ApConnectIPC` 91
`DR_ApDescribeBuffer` 92
`DR_ApDescribePnt` 94
`DR_ApInitIPC` 96
`DR_ApListBuffers` 97
`DR_ApListPoints` 99
`DR_ApPointBufAddress` 101
`DR_ApReadBlock` 103
`DR_ApReadControl` 105
`DR_ApReadPoint` 107
`DR_ApReadStatus` 109
`DR_ApUpdateBuffers` 111

DR_ApWriteBlock 112
DR_ApWriteControl 114
DR_ApWritePoint 116
V. Cascade Historian Functions 117
HI_Add 118
HI_BufferIDDestroy 120
HI_BufferIDLength 121
HI_BufferIDRead 122
HI_Bufsize 124
HI_ClipBuffer 126
HI_Count 127
HI_Deadband 128
HI_Delete 131
HI_Describe 132
HI_Disable 134
HI_Earliest 135
HI_Enable 136
HI_ExchangeBuffer 137
HI_FileBase 138
HI_Flush 140
HI_GapCountBuffer 141
HI_GapFillBuffer 142
HI_History 143
HI_Interpolate 145
HI_InterpolateData 148
HI_Latest 150
HI_Length 151
HI_List 152
HI_ScaleBuffer 154
HI_StatBuffer 155
HI_Register 156
HI_Unregister 157
HI_Version 158
VI. Inter-Process Communication Functions 159
IP_AddFDHandler 162
IP_AttachPhoton 163
IP_AttachPhotonMainloop 164
IP_ConnectToPort 165
IP_ConnectToService 166
IP_DetachPhotonMainloop 167
IP_GetChannelID 168
IP_GetConnectionID 169
IP_IsPulse 170
IP_ListenToPort 171
IP_ListenToService 172
IP_MsgCascade 173
IP_MsgCreate 174
IP_MsgData 175
IP_MsgDefaultSize 176
IP_MsgDestroy 177
IP_MsgInfoReply 178

IP_MsgInfoReplyRaw 179
IP_MsgLisp 180
IP_MsgRaw 181
IP_MsgRawData 182
IP_MsgResize 183
IP_NserveAdd 184
IP_NserveClose 185
IP_NserveInit 186
IP_NserveInitMyself 187
IP_NserveLookup 188
IP_NserveLookupId 189
IP_NserveLookupName 190
IP_NservePackTaskInfo 191
IP_NserveQueryNameCount 193
IP_NserveQueryNames 194
IP_NserveReattach 195
IP_NserveRemove 196
IP_NserveSetDomain 197
IP_pfTaskComp 198
IP_PhotonGUIFilter 199
IP_PhotonGUIHandler 200
IP_ProcessMessage 201
IP_PulseDestroy 202
IP_PulseNew 203
IP_PulseTimed 204
IP_PulseTrigger 205
IP_QueueClose 206
IP_QueueOpen 207
IP_QueueRead 208
IP_QueueStrerror 209
IP_QueueWait 210
IP_QueueWrite 212
IP_Receive 213
IP_ReceiveNonblock 215
IP_RemoveFDHandler 216
IP_Reply 217
IP_ReplyRaw 218
IP_SelectFD 219
IP_SetChannelID 220
IP_SetConnectionID 221
IP_SetGUIHandler 222
IP_TaskCloseAsync 223
IP_TaskCloseSync 224
IP_TaskConnect 225
IP_TaskCopy 226
IP_TaskCreate 227
IP_TaskCreateMe 228
IP_TaskDefaultDomain 229
IP_TaskDestroy 230
IP_TaskFindID 231
IP_TaskFindName 232

IP_TaskInitAsync 233
IP_TaskInitAsyncWrites 234
IP_TaskIntern 235
IP_TaskNew 236
IP_TaskSendAsync 237
IP_TaskSendRaw 238
IP_TaskSendSync 239
IP_TaskSetDomain 240
IP_TaskSetInfo 241
IP_TaskSetQname 243
IP_TaskUnintern 244
IP_TaskWaitAsync 245
IP_TaskZero 246
IP_TimerTime 247
IP_UnselectFD 248
VII. Cascade TextLogger Functions 249
LG_Cache 250
LG_Collect 252
LG_Disable 254
LG_Empty 255
LG_Enable 256
LG_Exit 257
LG_Fall 258
LG_File 260
LG_Flush 262
LG_Group 263
LG_Log 264
LG_Output 265
LG_Time 266
LG_Timestamp 268
LG_Tolerance 270
LG_UseGMT 271
VIII. Point Manipulation Functions 272
PT_FindCPoint 273
PT_InitClient 274
PT_NewCPoint 275
PT_PointCopyValue 276
PT_PointFormat 277
PT_PointInt 278
PT_PointReal 279
PT_PointString 280
Index ..... **??**
Colophon **284**

List of Tables

6-1. Cascade Historian Commands and Functions ..... **??**
6-2. Cascade Historian Binary Data Buffer Functions ..... **??**

I. Programmers Manual

Table of Contents

1. Introduction 1
2. Point Structure, Storage, and Manipulation 6
3. Interprocess Communication 8
4. The Cascade NameServer 13
5. Communicating with the Cascade DataHub 15
6. The Cascade Historian 30
7. Cogent Driver Specifications 32
A. GNU General Public License 35
B. GNU Lesser General Public License 41

Chapter 1. Introduction

The Cogent C API is an easy-to-use set of functions that fall into five main categories:

- Point structure, storage, and manipulation.
- Inter-process communication (synchronous and asynchronous).
- Global name server registration and lookup.
- Cascade DataHub reading, writing and exception reporting.
- Device driver control.

The Cogent Utilities are used by several Cogent products, primarily to facilitate inter-process communication.

To use the Cogent C API, you should be familiar with C programming.

1.1. When to use the different Cogent APIs

There are four Cogent APIs, grouped as:

- DataHub APIs for C++, Java, and .NET
- Cogent C API

The Cogent C API

This API lets you write high-speed clients that can interact with the Cascade DataHub, Cascade Historian, Cascade TextLogger, CIF Driver, DVN Driver, PFB Driver, and Gamma. It works in Linux, QNX 6, and QNX 4. Interprocess communication relies on Send/Receive/Reply message passing. In Linux, this is supported by Cogent's SRR Module. In QNX, this is supported by QNX's own Send/Receive/Reply protocol.

The DataHub APIs for C++, Java, and .NET

These three APIs share, as much as possible, common methods and syntax. For this reason they are distributed in one package and documented in a single book.

- **The DataHub API for C++** lets you write programs in C++ that connect to the DataHub over TCP, namely LAN, WAN, or the Internet.
- **The DataHub API for Java** lets you write programs in Java that connect to the DataHub over TCP, namely LAN, WAN, or the Internet. In addition, it lets you create web browser applications that receive and display live data from the DataHub.
- **The DataHub API for .NET** lets you write programs in .NET that connect to the DataHub over TCP, namely LAN, WAN, or the Internet. This API is implemented in C#, but can be used with any .NET language.

The following table shows the availability and support for these APIs in Windows, Linux, and QNX:

Language	Windows	Linux	QNX 6	QNX 4
C++	supported	unsupported	unsupported	unsupported

Language	Windows	Linux	QNX 6	QNX 4
Java	supported	unsupported	unsupported	not available
.NET	supported	may be available using Mono	not available	not available

For more information on these APIs, please refer to the DataHub APIs for C++, Java, and .NET manual.

1.2. Function Naming Conventions

The naming convention in these libraries consists of a two-letter package identifier followed by an underscore and a function name in which each word is capitalized. There are no separators between words within the function name.

- `DH` stands for the Cascade DataHub.
- `DR` stands for the CIF Driver.
- `HI` stands for the Cascade Historian.
- `IP` stands for Inter-Process communication.
- `PT` stands for Cascade DataHub points, and also for their data types.

1.3. System Requirements

QNX 6

- QNX 6.1.0 or later.
- GNU C.

QNX 4

- QNX 4.23A or later.
- Watcom C 10.6.

Linux

- Linux 2.4 or later.
- The GNU C compiler (GCC).
- The SRR IPC kernel module, which includes a synchronous message passing library modeled on the QNX 4 send/receive/reply message-passing API. This module installs automatically, but requires a C compiler for the installation. You can get more information and/or download this module at the Cogent Web Site.

1.4. Download and Installation

You can download the Cogent C API from the Cogent Web Site, and then follow these instructions for installing it on your system.

Cogent software comes packaged in self-installing archives available for download, or on diskette for commerically-licensed packages. Each software package name, which we refer to in these instructions as *`software_package_name`*, contains the product name, version number, operating system and sometimes other information, and will end with either `.sh.gz` or `.qpr`. For example, `gamma-4.0-bin-48-Linux.sh.gz` or `CascDataHub-4.0-bld10-x86-Cogent.qpr` are typical package names. The installation procedure is standardized across Cogent products, but depends on the operating system.

1.4.1. QNX 4

Option A: Install the archive from diskette.

1. Log in as root.
2. Insert the program diskette into your QNX 4 computer.
3. Type the command: **`install`**

   and respond to the system prompts.

Option B: Install the archive from a download or received as an e-file.

1. Download or copy the *`software_package_name`*`.sh.gz` file onto your QNX 4 computer.
2. Log in as root.
3. Type the command: **`gunzip`** ***`software_package_name`*****`.sh.gz`**

   This unzips the software package, and removes the `.gz` extension from the end of the filename.
4. Type the command: **`sh`** ***`software_package_name`*****`.sh`**

   and respond to the system prompts.

   If you get an error trying to install the `.sh` archive in QNX, please read the Installing program archives in QNX section of the Glossary, FAQ and Troubleshooting for help.

1.4.2. QNX 6

Option A: Use the QNX 6 Installer program. The Cogent repository is located at http://developers.cogentrts.com/repository.

Option B: Download the *`software_package_name`*`.qpr` file using the QNX 6 Voyager browser. The archive will install automatically.

Option C: Download or copy from diskette the *`software_package_name`*`.qpr` file onto your QNX 6 computer. Then (as root) run the command:

```
qnxinstall software_package_name.qpr
```

and respond to the system prompts.

1.4.3. Linux

First make sure the SRR kernel module is installed. If not, it is downloadable from the SRR for Linux page of the Cogent web site. Then follow these instructions to install the software package:

1. Download or copy from diskette the *software_package_name*.sh.gz file onto your Linux computer.
2. Log in as root.
3. Type the command: **gunzip *software_package_name*.sh.gz**

   This unzips the software package, and removes the .gz extension from the end of the filename.
4. Type the command: **sh *software_package_name*.sh**

   and respond to the system prompts.

1.4.4. Installed file locations

On whichever OS the software is installed, all files will be written to the /usr/cogent/ directory. Depending on which packages are installed, the following subdirectories will contain the types of files shown:

bin/	Binary executables.
dll/	Dynamically-linked libraries.
docs/	Miscellaneous documentation. (Regular documentation is downloaded separately.)
include/	Header files.
lib/	Cogent library files.
license	The license file (see below).
require/	Lisp or Gamma files used by Gamma or its extensions.
src/	The source code for examples, tests, tutorials, etc.

1.4.5. Installing licenses

Licenses to use the software can be purchased from Cogent. To install a license, you need to copy the license string into the /usr/cogent/license file. If this file does not exist on your system, just create one as a text file and list the license strings, one per line.

If a license is not installed, you will see the following console message on startup:

```
software_package_name: No valid licenses found.
This program is running in demo mode and will terminate after 1 hour.
```

and the software will run for one hour in demo mode.

1.5. Cogent Product Integration

Cogent products work together to support real-time data connectivity in Windows, Linux, and QNX. They can be dynamically integrated as a group of modules where each module connects to any other module(s) as needed. New modules can be added and existing modules reconfigured or modified, all during run-time. Data in any module of the system can be collected and redistributed to any other module via the Cascade DataHub and Cascade Connect. Communication with field devices is provided by one of several Cogent Device Drivers. Historical records of unlimited size can be maintained and queried with the Cascade Historian, and ASCII text files can be logged with the Cascade TextLogger.

Custom programs written in C or C++ can interface with the system, using the Cogent C API or the DataHub APIs for C++, Java, and .NET. In addition, Cogent's own dynamically-typed object-oriented programming language, Gamma, is fully compatible with all modules. User interfaces can be created in Gamma, which supports Photon in QNX and GTK in Linux.

1.6. Where can I get help?

If you are having problems with a Cogent product, first check the Troubleshooting Guide. If you can't find the answer there, you can contact Cogent Real-Time Systems, Inc. for technical support for any product you have purchased.

- Email: <`support@cogent.ca`>
- Phone: 1-888-628-2028
- Fax: (905) 702-7850

Chapter 2. Point Structure, Storage, and Manipulation

The Cascade DataHub represents points given the following properties:

- **Point type:** One of integer, floating point, or character string.
- **Point value:** Based on the point type.
- **Confidence factor:** 0 - 100 percent, unused by most applications.
- **Name:** The character string representing the point name.
- **Locked:** `0` for locked, or `1` for unlocked.
- **Security:** `0` to `32768`, where higher numbers represent higher security.
- **Address:** An internal pointer carrying the DataHub address where the point was last found.
- **User Data:** A void pointer which can be used to store user-defined data with a point structure. In addition, the Cascade DataHub uses other point attributes internally, which are not available to the user. These deal with the clients who have registered for exceptions, and the necessary flags to ensure that all clients receive the latest values for all points, regardless of communication link speed.

The Cascade DataHub point structure is called `PT_stCPOINT`, and a convenience pointer to the structure type is called `PT_pCPOINT`. The Cogent C API provides a number of functions for manipulating the `PT_stCPOINT` structures.

2.1. Creating Points

There are two ways to "create" a new data point: create a new point structure, or change the `name` of an existing point structure.

Creating a point structure is done either by creating a local variable of type `PT_stCPOINT`, or by allocating it on the heap (through **malloc**). The function `PT_NewCPoint` will allocate a point on the heap, and initialize all of its internal values to sensible numbers. This function is useful when you would like to maintain all of the point definitions internally to your program, and would like to manage the storage and lookup of these points yourself. The `PT_FindCPoint` function will automatically call `PT_NewCPoint` if it attempts to look up a point that does not currently exist in your application.

Changing the `name` of a point (or `PT_stCPOINT` structure) means changing the value of the `name` field. When doing this, the `address` field must be set to `0`.

Zeroing an entire point structue (ie. setting all field values to `0`) for points of type `PT_TYPE_STRING` will create a memory leak.

2.2. Maintaining a Point Hash Table

The Cogent third-party library provides a mechanism for internally managing DataHub point information within your own program. This mechanism uses a hash table based on the name of the point. The hashing function uses binary search in order to resolve collisions within the hash table. The maximum length of the ordered array is 32000 points, and the length of the hash table is 256 entries. This places a limit of 32000 points per hash entry and 8192000 total points in the table. At the time of this writing, the hashing function uses a first-character lookup on the point name, meaning that the hash table preserves alphabetical order, but imposes a limit of 32000 points starting with the same letter on the datahub.

In order to maintain a point hash table within your program, you must use the function `Pt_InitClient`, which will set up a static hash table within your program. In order to look up a point within the hash table, you use the function `PT_FindCPoint`, which will create an entry in the hash table if necessary and return the resulting point structure.

2.3. Accessing and Copying Point Values

Cascade DataHub points may take on one of three types:

- integer, using the type `PT_TYPE_INT32`;
- floating point, using the type `PT_TYPE_REAL`; and
- string, using the type `PT_TYPE_STRING`.

Since the type may change during the life of a point, care must be taken to ensure that the appropriate value is used in a given situation. This can be simplified by using the accessor functions `PT_PointString`, `PT_PointInt` and `PT_PointReal`, which will perform a best-guess conversion of the point type to the type requested. The `PT_PointString` function will also format the point value according to a printf-style format directive. The value of a point really includes attributes like its timestamp, confidence factor and security, so it is better to copy the point value using the `PT_PointCopyValue` function, which will copy all related information into the destination point. If the type of the point is `PT_TYPE_STRING`, then the point value will be allocated on the heap. In addition, if the destination point is an existing point of type `PT_TYPE_STRING`, then its current value will be freed using the `free()` C library call.

2.4. Memory Allocation and String Values

String values for points are normally allocated on the heap. Some functions, notably `PT_PointCopyValue`, make this assumption and may free the string pointer or allocate space for a new value. This is not true for a point name. The programmer must control the memory associated with the point name in the `PT_stCPOINT` structure.

Zeroing an entire point structue (ie. setting all field values to `0`) for points of type `PT_TYPE_STRING` will create a memory leak.

Chapter 3. Interprocess Communication

This package is written to be source code compatible across the QNX 4, QNX 6 and Linux operating systems. In Linux, the SRR Module is required.

In addition to this library, two programs are required: **qserve** and **nserve**. These should have been included with this package or another sofware package from Cogent. **qserve** provides asynchronous message queueing services, and offers a number of small changes from the semantics of the POSIX **mqueue** program. The major differences are the semantics of message notification, transparent networking, and the removal of a queue when its creating process terminates. **nserve** provides name resolution services on a per-process basis, allowing processes to easily identify one another by name, thereby making it possible for them to communicate. **nserve** is intended to maintain consistent name information across a network. In addition, **nserve** can inform all processes on the network via **qserve** whenever a named process start or stops.

Node ID has different meanings in QNX 4, QNX 6 and Linux. In QNX 4, the node ID is the node number, which is unique on the network, and always refers to the same node. In QNX 6, the node ID is the "node descriptor", which is only unique on a single machine, and is transient on the network. The node ID, when valid, indicates the identifier of the foreign node as seen by the current node. A node ID of zero always indicates the node on which the process is running. In Linux, the node ID is currently meaningless, as the SRR Module does not implement network message passing.

3.1. Connections and Channels

Connection and channel identifiers and node names are only relevant to QNX 6. Under QNX 4 and Linux, connection and channel should always be zero, and node name will either be the string representation of a node number, or "".

- `IP_SetConnectionID` - sets the connection ID for IP library use.
- `IP_GetConnectionID` - returns the connection ID for IP library use.
- `IP_SetChannelID` - sets the channel ID for IP library use.
- `IP_GetChannelID` - returns the channel ID for IP library use.

3.2. Task Structure Caching

Some applications find it useful to cache task structures in an easily accessible table in the local application, rather than making a query to the name server every time task information is required. This caching is called "interning" the task structure. A task structure can be "found" locally only if it has been interned.

- `IP_TaskFindName` - finds a task by its name.
- `IP_TaskFindID` - finds a task by its node, process, and channel IDs.
- `IP_TaskCopy` - copies a task structure.
- `IP_TaskIntern` - adds a task to a process’s task cache.
- `IP_TaskUnintern` - removes a task from a process’s task cache.

3.3. Messages

Message structures are used to encapsulate the information passed between processes. The underlying messaging primitives do not specify a data format so information concerning the sending task, the message length and message type is not necessarily available. The `IP_Msg` structure efficiently encapsulates this information, while following the OS convention on message type and subtype header information.

An `IP_Msg` is in fact a wrapper on a data buffer. Only the `msg` portion is transmitted or received into during an interprocess communication transaction. In most cases, the `msg` portion is a pointer to an `IP_MsgBuffer` structure which defines the message type, subtype, length and identifying information about the sender. The `IP_MsgBuffer` structure is of varying length, with the last part of the structure containing the data.

- `IP_MsgDefaultSize` - gets the default size of interprocess messages.
- `IP_MsgCreate` - creates an `IP_Msg` structure.
- `IP_MsgData` - returns a pointer to the data payload of an `IP_Msg` structure.
- `IP_MsgDestroy` - frees memory associated with a message.
- `IP_MsgResize` - resizes the `IP_MsgBuffer`, if possible.
- `IP_MsgCascade` - writes message data to an `IP_MsgBuffer`.
- `IP_MsgRaw` - like `IP_MsgCascade`, with `IP_SUB_RAW` for its *`subtype`*.
- `IP_MsgRawData` - gives a pointer to `IP_RAW` message data.
- `IP_MsgLisp` - constructs a formatted text message.

3.4. Cascade NameServer Functions

The Cascade NameServer (**nserve**) is a necessary part of a system using the `IP_*` functions. Its purpose is to map a node, a process ID, and (in QNX 6) a channel ID, to a process name. A program that uses the Cogent `IP_*` library functions will normally declare a name with **nserve** in order to advertise its presence. **nserve** also carries a queue name for asynchronous communication, and a Cascade DataHub domain name where applicable.

For more information on **nserve** see: The Cascade NameServer chapter of this manual.

- `IP_NserveInit` - creates a task structure and informs **nserve**.
- `IP_NserveInitMyself` - declares current task structure information to **nserve**.
- `IP_NservePackTaskInfo` - makes a Lisp-parseable version of task information.
- `IP_NserveLookup` - fills in a known task structure.
- `IP_NserveLookupId` - finds a task by node, process and channel ID.
- `IP_NserveLookupName` - allocates and fills in a new task structure.
- `IP_NserveAdd` - adds an entry to **nserve**.
- `IP_NserveRemove` - removes an entry from **nserve**
- `IP_NserveReattach` - closes and renews all task connections and queues.
- `IP_NserveClose` - closes the channel to **nserve**.
- `IP_NserveSetDomain` - changes a task's domain name.

- `IP_NserveQueryNameCount` - gives the number of registered names.
- `IP_NserveQueryNames` - fills an array with **nserve**'s names.

3.5. Photon Functions

Programs using the Photon® microGUI will operate in one of two modes: a) allowing the Photon function, `PtMainLoop`, to implement the event loop, or b) by implementing an event loop using `IP_Receive`. Different set-up calls are required depending on which of these methods is used. It is possible to set up a program to use both of these methods, at the user's option.

- `IP_DetachPhotonMainloop`
- `IP_AttachPhotonMainloop`
- `IP_AttachPhoton`
- `IP_PhotonGUIFilter`
- `IP_PhotonGUIHandler`

3.6. Pulses and Timers

A *pulse* is an asynchronous message whose primary purpose is to signal to a process that an event has occurred. The occurrence of the pulse is the important information, not the payload that the pulse might carry. In the Cascade `IP_*` functions, a pulse cannot carry a payload, so that its only information content is its occurrence. Pulses in QNX 4 and Linux are implemented as proxies.

Timers can be attached to pulses such that the pulse is delivered when the timer expires, or at a set interval. This library does not deal with timers that deliver operating system signals. Using signals for asynchronous messaging is generally considered poor practise.

- `IP_PulseNew` - creates a new pulse.
- `IP_PulseDestroy` - destroys a pulse.
- `IP_PulseTimed` - sets up timers to trigger pulses.
- `IP_IsPulse` - validates a received message against a pulse ID.
- `IP_TimerTime` - adjusts pulse timer parameters.
- `IP_PulseTrigger` - immediately sends a pulse.

3.7. Cascade QueueServer Functions

The low-level interface to the Cascade QueueServer (**qserve**) is normally not used directly by a program. These functions are entirely encapsulated within other functions in this library. They are included here for completeness.

- `IP_QueueOpen` - opens a queue for reading or writing.
- `IP_QueueClose` - closes a queue.
- `IP_QueueWrite` - writes a message to a queue.
- `IP_QueueRead` - reads a message from the queue.
- `IP_QueueWait` - requests notification of an event.

- `IP_QueueStrerror` - gives access to error strings.

3.8. Receiving Messages and Events

The Cogent `IP_*` functions unify the process event space by treating all forms of input and notification as events.

- `IP_SetGUIHandler` - sets callback functions for GUI events.
- `IP_Receive` - receives any message.
- `IP_ReceiveNonblock` - receives any message, without blocking.
- `IP_AddFDHandler` - tells IP_Receive to accept file input.
- `IP_RemoveFDHandler` - prevents IP_Receive from accepting file input.
- `IP_SelectFD` - is used internally only.
- `IP_UnselectFD` - is used internally only.
- `IP_ProcessMessage` - classifies messages for IP_Receive.

The `IP_Receive` function can accept input from the following sources:

1. a message from another process using the Cogent `IP_*` functions
2. a message from another process using the OS IPC primitives
3. an asynchronous message via Cascade QueueServer
4. data available on a file descriptor
5. a pulse
6. a signal
7. a Photon microGUI event
8. a task death notification for `_PPF_INFORMED` tasks (QNX 4 and Linux)

3.9. Replying to Messages

Some messages and events require a reply. The reply can be via the low-level interprocess communication mechanism, or indirectly via an `IP_MsgInfo` structure. This structure is filled by `IP_Receive` with enough information to facilitate a reply. Further, the message can be either a Cascade IPC message structure (`IP_Msg`), or it can be a raw binary buffer. In the case of a raw reply, it is the programmer's responsibility to ensure that the message is correctly formatted for the receiver. `IP_MsgInfoReply` is the normal means of replying.

- `IP_Reply` - replies to an IP_SYNC message using rcvid.
- `IP_MsgInfoReply` - replies to an IP_SYNC message using IP_MsgInfo.
- `IP_ReplyRaw` - replies to an IP_RAW message using rcvid.
- `IP_MsgInfoReplyRaw` - replies to an IP_RAW message using IP_MsgInfo.

3.10. Sending Messages

A task may send messages either synchronously or asynchronously. With a synchronous message, the sender must block waiting for the receiver to reply. There is no time limit on how long the sender may block. With an asynchronous message, the sender transmits the message to a queue, and immediately returns without blocking. The receiver is notified that a message is waiting, and gets it at some later time through a call to *`IP_Receive`*.

- `IP_TaskConnect` - opens a connection to a receiver.
- `IP_TaskSendSync` - transmits a message, and waits for a reply.
- `IP_TaskSendAsync` - transmits a message via **qserve** and returns immediately.
- `IP_TaskSendRaw` - sends data in bytes, synchronously.

3.11. Task Structures

- `IP_TaskCloseAsync` - closes queues and cleans up resources.
- `IP_TaskCloseSync` - closes synchronous connections and cleans up resources.
- `IP_TaskSetInfo` - sets the fields in a task structure.
- `IP_TaskCreate` - creates a new task structure.
- `IP_TaskCreateMe` - creates a task structure for the current process.
- `IP_TaskDefaultDomain` - returns the Cascade DataHub default domain.
- `IP_TaskDestroy` - closes connections and queues, removes the task and frees memory.
- `IP_TaskInitAsync` - creates a read-only queue.
- `IP_TaskInitAsyncWrites` - opens a task's queue as write-only.
- `IP_pfTaskComp` - compares two tasks for equality.
- `IP_TaskWaitAsync` - registers the task for events in **qserve**.
- `IP_TaskZero` - sets all task structure fields to defaults.
- `IP_TaskNew` - creates a new task structure.
- `IP_TaskSetDomain` - sets or changes a task's domain name.
- `IP_TaskSetQname` - sets a task's queue name.

3.12. Working with TCP/IP

This library provides some simple functions for implementing TCP/IP servers and clients. These functions are primarily provided to simplify the interface with the `IP_Receive` function. These functions always create an Internet Domain socket, and always assume a TCP connection as opposed to UDP or ICMP.

- `IP_ListenToPort` - is a wrapper for `listen`.
- `IP_ListenToService` - like `IP_ListenToPort`, but uses the service name.
- `IP_ConnectToPort` - resolves a host name and connects to a port.
- `IP_ConnectToService` - like `IP_ConnectToPort`, but uses the service name.

Chapter 4. The Cascade NameServer

The Cascade NameServer (**nserve**) provides network name services, information about queues and domains for all registered tasks, and alert messages to registered tasks whenever another task starts or stops on the network. The name server must be started after the Cascade QueueServer, **qserve**, usually through the `sysinit.N` startup file. Both of these utilities are started automatically by most Cogent products on startup, if they are not running already.

nserve is used to maintain information about currently running tasks in the Cascade DataHub universe. All names registered on the name server are considered global within the system network, and all names should be unique on the network. This facility cannot be used as a global semaphore due to the race conditions associated with more than one task attempting to use the same name. Tasks whose names do not particularly matter can be made up of a combination of node number and task id.

When a task starts up, it must register a name, queue, and domain with the name server in order to make itself available to other tasks in the system. A task is not strictly obligated to have a queue name, but it will not be able to receive asynchronous messages without one. This initialization is performed through the `IP_NserveInit` function. The domain of a task may be changed at any time using the `IP_TaskSetDomain` function.

QNX Note: **nserve** provides facilities not available through the QNX **nameloc** program. It uses **nameloc** to identify itself to the QNX operating system and any Cascade DataHub processes, but the Cascade DataHub processes themselves do not need to register with **nameloc**. The **nserve** facility extends **nameloc** with:

- Asynchronous notification of tasks starting and stopping on a network-wide basis, without the need for a task to become "informed."
- Maintenance of a queue name for each task, so tasks can communicate asynchronously.
- Maintenance of a domain name for each task, so you can keep track of tasks' peers throughout the network.

4.1. Domains

All tasks using **nserve** must declare a domain. This is a character string of up to 15 characters that differentiates groups of cooperating processes. All processes with the same domain name are considered to be running in the same domain. This differentiation generally only affects the manner in which processes retrieve information from the Cascade DataHub.

Tasks do not need to specify a domain name for points in the DataHub running in that domain. In order to access points in another domain, a process must prepend the domain name followed by a colon to the point name. Similarly, the Cascade DataHub will transmit a point name with the domain name attached when delivering an exception to a task in another domain. For example, if a task is running in the default domain and wishes to read a point, `water_level` in the `control` domain's DataHub, it must access that point as `control:water_level`.

4.2. Locating Other Tasks on the Network

Any task which is registered through **nserve** can be located through a call to one of these three functions:

- **`IP_NserveLookup`** finds a task by name and has **nserve** fill in its structure.

- **`IP_NserveLookupID`** finds a task by node, process and channel ID, and has **nserve** fill in its structure.
- **`IP_NserveLookupName`** takes a name registered with **nserve** and fills in a new task structure for it.

4.3. Task Started and Stopped Messages

nserve automatically transmits a message to all tasks that have a queue whenever another task registered with **nserve** starts or stops. This mechanism makes it very simple for user tasks to respond to changes in the system. For example, a user task may want to know when a DataHub for a particular domain has started in order to register for exceptions from some or all of its points. These messages will be transmitted to the user task with a message type of `IP_ASYNC` and a command of `IP_SUB_LISP`. When a task starts or stops, **nserve** will send a message to all other registered tasks on the network as: `(taskstarted name queue domain node pid)`, or `(taskdied name queue domain node pid)`.

Chapter 5. Communicating with the Cascade DataHub

The Cascade DataHub contains a snapshot of the current values of all of its points. A point is essentially a name and an associated value, along with auxiliary information such as time stamp, security and lock status. The following functions provide access to the Cascade DataHub through the Cascade DataHub IPC functions (`IP_*`).

The Cascade DataHub provides three basic data services:

- **Read** A user task synchronously reads a point value from the DataHub. See `DH_ReadPoint` and `DH_ReadExistingPoint`.
- **Write** A user task synchronously writes a point to the DataHub. See `DH_WritePoint` and `DH_WriteExistingPoint`.
- **Exception** A user task informs the DataHub that it would like to be alerted whenever a point value changes, and to have an unsolicited (asynchronous) message sent to it. See `DH_RegisterPoint`, `DH_RegisterExistingPoint`, `DH_RegisterAllPoints`, `DH_ParsePointMsg`, and `DH_ParsePointString`.

5.1. Exceptions

The Cascade DataHub is designed using the exception paradigm of point value transmission. That is, a client task can tell the DataHub that it would like to be informed whenever a value changes on one or more points, and then it simply waits for the DataHub to transmit the changes. This mechanism generally results in less network traffic and substantially reduced delays compared with the more popular polling method of DataHub query. The Cascade DataHub automatically concatenates point messages to be sent to a client if more than one exception occurs before a message is sent out.

5.2. Echoes

In an exception-based system, it is possible for a task to both write a point value in the datahub, and receive exceptions on that point. This creates a potential positive feedback situation where a task will essentially start talking to itself forever through the DataHub. Even in cases where the client task chooses to ignore point values that have not changed since the last message, it is still possible to get more than one value for a point into the communication "pipeline" and generate the same infinite loop. The Cascade DataHub solves this by marking exceptions as either a normal exception, or as an echo. An echo is a point exception which is being returned to the task which originally transmitted the value change to the DataHub. Most user tasks should ignore or treat echo messages differently from regular exception messages.

5.3. Non-Existent Cascade DataHub Points

The Cascade DataHub, by default, creates any points that do not currently exist whenever it receives a read, write or exception request message. This allows the DataHub to populate itself without having any a-priori information about the system in which it is running. On occasion, it may be useful for a task to determine whether a point exists. This may be done using the functions `DH_WriteExistingPoint`, `DH_ReadExistingPoint` and `DH_RegisterExistingPoint`. These functions will return an error if the named point does not exist when the function is called.

5.4. Parsing Point Messages

When an exception arrives from the DataHub, it is in a form that must be parsed in order to extract the type, value, timestamp, etc. This can be done easily with the functions `DH_ParsePointMsg` and `DH_ParsePointString`.

5.5. Optimizing Throughput

To optimize data throughput between clients and the DataHub, custom clients can make two calls:

1. `DH_SetTransmitFormat(msgformat);`
2. `DH_SetReceiveFormat(htask, hmsg, domain, msgformat, NULL);`

msgformat is one of:

- `PT_FMT_ASCII`
- `PT_FMT_BINARY`

DH_SetTransmitFormat instructs the API to transmit all point change messages from the custom client to the DataHub using the specified message format.

DH_SetReceiveFormat instructs the DataHub to transmit all messages to the custom client using the specified format. This call will fail with `ST_NO_TASK` if the custom client has never successfully called one of DH_RegisterPoint, DH_RegisterExistingPoint or DH_RegisterAllPoints. This is because the Cascade DataHub does not maintain internal state information for clients that are not registered to receive point exceptions.

5.6. Point Size Limit

There is an inter-process communication message size limit that all Cogent processes must agree upon. This defaults to 8192 bytes, and no point value can exceed this number, less a certain variable amount of header, so conservatively this number is about 8100 bytes. We'll call this the *IPC Message Size*.

There is an additional buffer specific to the DataHub that used to create exception messages which is nominally 1000 bytes in length. This is kept small to limit the number of exceptions that go out in a single IPC message. We'll call this the *Exception Message Size*.

When a point representation is constructed, it is constructed in the exception buffer, so that limit is 1000 bytes, less some header. A string value is further reduced because strings may contain escaped characters, and the rendering code estimates the size conservatively by assuming the worst case: all characters are escaped, and each character in the input string requires two characters to render. This limits the string to (1000 - header) / 2, giving us a few characters less than 500.

You can increase the Exception Message Size with the `-b` option to the DataHub. This size cannot be greater than the IPC Message Size. There is also a hard limit of 65000 characters.

You can increase the IPC Message Size by setting the environment variable `IP_MSG_DEFAULT_SIZE`. You *must* make this environment variable the same for all programs that are going to communicate with the DataHub. This size cannot be greater than 65535 if you plan to have it run in Linux, or over a QNX network.

For example, to work with larger strings:

```
[sh]$ export IP_MSG_DEFAULT_SIZE=65535
...
[sh]$ datahub -b 65000
...
```

```
[sh]$ writept -s test '1234567890...' ❶
...
[sh]$ readpt -s test
```

❶ Insert a large number of characters here.

These settings would allow you to create strings up to approximately (65000 - 30) / 2 bytes in the DataHub.

5.7. Cascade DataHub API Code Examples

These code examples are provided in source form as part of the Cascade DataHub API, normally installed in the `/usr/cogent/src/datahub` directory. They are provided here so that you can see what is involved in communicating with the DataHub using your own C programs. The Gamma programming language also contains many 'hooks' into the DataHub. For more information about using Gamma with the DataHub see the DataHub demo program that is available for download from the Cogent web site. The Cascade DataHub demo uses Gamma to show the features of the DataHub. If you download and run the Gamma demo for the DataHub you will have to reinstall your commercial version of the DataHub, because the version that comes with the Gamma demo contains a time limited license.

5.7.1. Reading from the Cascade DataHub

```
/*
 * Cascade DataHub point reader:  readpt
 *
 * (C) Copyright Cogent Real-Time Systems Inc., 1997.  All rights reserved.
 *
 * This program reads a point from the Cascade DataHub and displays the
 * result on the standard output.
 *
 * This program is supplied with the Cascade DataHub programming API.  It
 * may be copied or modified, in whole or in part, for the sole purpose of
 * creating applications to be used with the Cascade DataHub.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>

#include <cogent.h>

/*
 * A point is one of three types.  We discover which one we have and
 * print the appropriate name, value, confidence factor, lock status
 * and security level.
 */

void print_point (PT_pCPOINT ppoint, int brief)
{
 char  *t;

 printf ("Point: %s\n", ppoint->name);

 switch (ppoint->type)
 {
  case PT_TYPE_INT32:
   printf ("Value: %ld\n", (long)ppoint->value.i);
   break;
  case PT_TYPE_REAL:
   printf ("Value: %g\n", ppoint->value.r);
   break;
```

```
  case PT_TYPE_STRING:
   printf ("Value: %s\n", ppoint->value.s);
   break;
  default:
   printf ("Value: ...unknown point type!\n");
   break;
 }
 if(!brief)
 {
  t = ctime ((time_t*)&ppoint->seconds);
  t[19] = '\0';
  printf ("Time:  %s.%03ld\n", &t[4],
    (long)ppoint->nanoseconds / 1000000);
  printf ("Conf:  %d\n", ppoint->conf);
  printf ("Lock:  %d\n", ppoint->locked);
  printf ("Secur: %d\n", ppoint->security);
 }
}

#ifdef __USAGE
Copyright (C) Cogent Real-Time Systems Inc., 1996-1997

%C [-b] [-d domain] pointname

   -b    brief output (only the point name and value)
   -d    domain, replaces the default

Read pointname from the Cascade datahub, the domain
can be specified with -d, or by qualifying the pointname,
as in "domain:thepointname" (the latter technique overriding
the former).
#endif

const char* UT_USAGE =
 "Usage: %s [-b] [-d domain] pointname\n"
 ;

const char* UT_HELP =
 "\n"
 "Help:\n"
 "    -b          Brief output (only the point name and value).\n"
 "    -d          Domain, replaces the default.\n"
 "\n"
 "Read pointname from the Cascade datahub, the domain\n"
 "can be specified with -d, or by qualifying the pointname,\n"
 "as in \"domain:thepointname\" (the latter technique overriding\n"
 "the former).\n"
 ;

int main (int argc, char** argv)
{
 IP_Msg   *hmsg;
 ST_STATUS  status;
 PT_stCPOINT  point;
 char   *ptname = NULL, *domain = NULL, *myname;
 int    opt;
 int    brief = 0;
 IP_Task   *htask;

 /*
  * Parse the input arguments.  The only interesting argument is an
  * alternate datahub domain name.  We really do not need this, as
  * we could specify the point name as domain:name
  */
 while ((opt = getopt(argc, argv, "hbd:")) != -1)
 {
  switch (opt)
  {
```

```
   case 'h':
    UT_Help(argv[0], UT_USAGE, UT_HELP);
  exit(0);

   case 'd':
  domain = optarg;
  if (strlen(domain) > 15)
   domain[15] = '\0';
  break;

   case 'b':
    brief = 1;
    break;

   default:
  UT_Usage (argv[0], UT_USAGE, stderr);
  exit(1);
 }
}

if (!argv[optind])
{
 UT_Usage(argv[0], UT_USAGE, stderr);
 exit(1);
}

/*
 * Initialize communication through the Cascade IPC library.  We do
 * not want other tasks to be notified of the start and stop of
 * this task, so we do not use IP_NserveInit.  The name server
 * will never know about this task, so notifications will not be
 * passed on.  Tasks that attempt to look up their clients in the
 * name server will treat this task as non-existent.  For example,
 * Cascade DataHub would not be able to send point exceptions to
 * this task.
 */

if ((myname = strrchr(argv[0], '/')))
 myname++;
else
 myname = argv[0];

if (!(htask = IP_TaskCreateMe (IP_GetChannelID(), myname, domain,
          NULL, 0)))
{
 fprintf (stderr, "Could not initialize Cascade IPC subsystem\n");
 exit (1);
}

/*
 * Create a pre-allocated message structure for use with all
 * IPC calls.  This includes the DH_* functions.  The API could have
 * created its own internal message structure, but this would have
 * left us with no way to control its size or be efficient about
 * allocation.  This way we do a little more work, but have more
 * control of what is being allocated.
 */
hmsg = IP_MsgCreate (NULL, IP_MsgDefaultSize(), 0);

while((ptname = argv[optind++]))
{
 /*
  * Zero the point structure.  If we do not do this, the address
  * field could be non-zero, and then the API will take that to be a
  * cached datahub address.  That might cause a crash.
  */
 memset (&point, 0, sizeof(point));
```

```
  /*
   * Provide a point name buffer separately from the rest of the point
   * structure.  There is no way for the API to know what the allocation
   * status of a point name is, so it will never attempt to free this
   * buffer, nor write into it.
   */
  point.name = ptname;

  if ((status = DH_ReadPoint (htask, &point, hmsg, NULL)) != ST_OK)
  {
   fprintf (stderr, "Read \"%s\" failed: %s\n",
    point.name, ST_StatusName (status));
   exit(1);
  }
  else
  {
   print_point (&point, brief);
  }
 }

 return 0;
}
```

5.7.2. Writing data to the Cascade DataHub

```
/*
 * Cascade DataHub point writer:  writept
 *
 * (C) Copyright Cogent Real-Time Systems Inc., 1997.  All rights reserved.
 *
 * This program writes a point to the Cascade DataHub.
 *
 * This program is supplied with the Cascade DataHub programming API.  It
 * may be copied or modified, in whole or in part, for the sole purpose of
 * creating applications to be used with the Cascade DataHub.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <errno.h>

#include <cogent.h>

#ifdef HAVE_SYS_TIME_H
#include <sys/time.h>
#endif

const char* UT_USAGE =
 "Usage: %s [-d domain] [-r|-f|-i|-l|-s] [-S security] pointname pointvalue\n"
 ;

const char* UT_HELP =
 "\n"
 "Help:\n"
 "    -d domain   Set the domain for the operation.\n"
 "    -r          Write as a real (floating point) number.\n"
 "    -i          Write as a short integer.\n"
 "    -l          Write as a long integer.\n"
 "    -s          Write as a character string.\n"
 "    -S          Set security level for write.\n"
 "\n"
 "Write a point to the Cascade datahub in the given domain.\n"
 "\n"
 "Notes:\n"
 "- Strings containing spaces and special characters must be escaped\n"
```

```
   "    from the shell appropriately.\n"
 "\n"
 "- The type will be guessed if not specified, the order of guessing is:\n"
    "    1) long\n"
    "    2) float\n"
    "    3) string (the default)\n"
   "    The guess is considered correct if the entire argument is converted.\n"
 "\n"
 "- A long can be given in standard C style, 0x5f (hex), 0647 (oct), etc.\n"
 ;

int main (int argc, char** argv)
{
 IP_Msg   *hmsg;
 ST_STATUS  status;
 PT_stCPOINT  point;
 char   *ptname = NULL, *ptvalue = NULL, *domain=NULL, *myname;
 short   type = PT_TYPE_VOID;
 int    security=0;
 IP_Task   *htask;
 int    opt;

 /*
  * Parse the command line
  */
 while((opt = getopt(argc, argv, "hd:rilsS:")) != -1)
 {
  switch (opt)
  {
    case 'h':
     UT_Help(argv[0], UT_USAGE, UT_HELP);
   exit(0);
    case 'd':
   domain = optarg;
   if (strlen(domain) > 15)
    domain[15] = '\0';
   break;
    case 'r':
   type = PT_TYPE_REAL;
   break;
    case 'i':
    case 'l':
   type = PT_TYPE_INT32;
   break;
    case 's':
   type = PT_TYPE_STRING;
   break;
    case 'S':
   security = atoi (optarg);
   break;
    default:
   UT_Usage (argv[0], UT_USAGE, stderr);
   exit (1);
   break;
  }
 }

 /* the last two args better be the point and value */

 if (!argv[optind] || !argv[optind+1])
 {
  UT_Usage(argv[0], UT_USAGE, stderr);
  exit(1);
 }
 ptname = argv[optind];
 ptvalue = argv[optind+1];

 /*
```

```
 * Initialize communication through the Cascade IPC library.  We do
 * not want other tasks to be notified of the start and stop of
 * this task, so we do not use IP_NserveInit.  The name server
 * will never know about this task, so notifications will not be
 * passed on.  Tasks that attempt to look up their clients in the
 * name server will treat this task as non-existent.  For example,
 * Cascade DataHub would not be able to send point exceptions to
 * this task.
 */

if ((myname = strrchr(argv[0], '/')))
 myname++;
else
 myname = argv[0];

if (!(htask = IP_TaskCreateMe (IP_GetChannelID(), myname, domain,
          NULL, 0)))
{
 fprintf (stderr, "Could not initialize Cascade IPC subsystem\n");
 exit (1);
}

/*
 * Set this task's security level.  This level must be greater than or
 * equal to the security level of the point in the datahub in order
 * for the write to succeed.  The datahub does not know whether this
 * task has the right to claim this security level.  That enforcement
 * is up to the programmer of the user task.
 */
IP_TaskSetSecurity (htask, security);

/*
 * Create a pre-allocated message structure for use with all
 * IPC calls.  This includes the DH_* functions.  The API could have
 * created its own internal message structure, but this would have
 * left us with no way to control its size or be efficient about
 * allocation.  This way we do a little more work, but have more
 * control of what is being allocated.
 */
hmsg = IP_MsgCreate (NULL, IP_MsgDefaultSize(), 0);

/*
 * Zero the point structure.  If we do not do this, the address
 * field could be non-zero, and then the API will take that to be a
 * cached datahub address.  That might cause a crash.
 */
memset (&point, 0, sizeof(point));

/*
 * Provide a point name buffer separately from the rest of the point
 * structure.  There is no way for the API to know what the allocation
 * status of a point name is, so it will never attempt to free this
 * buffer, nor write into it.
 */
point.name = ptname;
point.type = type;
point.conf = 100;

/*
 * Set the time on the point.  If this is not set, then the datahub
 * will show a zero time.
 */
#ifdef __QNX__
{
 struct timespec tp;

 clock_gettime (CLOCK_REALTIME, &tp);
 point.seconds = tp.tv_sec;
```

```
  point.nanoseconds = tp.tv_nsec;
 }
#else
 {
  struct timeval tp;
  gettimeofday (&tp, NULL);
  point.seconds = tp.tv_sec;
  point.nanoseconds = tp.tv_usec * 1000;
 }
#endif /* __QNX__ */

 /*
  * Set the value of the point based on the type.
  */
 switch (point.type)
 {
   case PT_TYPE_INT32:
    point.value.i = strtol(ptvalue, 0, 0);
  break;

   case PT_TYPE_REAL:
  point.value.r = strtod (ptvalue, NULL);
  break;

   case PT_TYPE_STRING:
  point.value.s = ptvalue;
  break;

   case PT_TYPE_VOID:
   default:
   {
    /* try to autodetect type of point */
    char* eos = 0;

    /* it's a long if conversion goes to end of string */
    eos = 0;
    point.value.i = strtol(ptvalue, &eos, 0);
    if(*eos == '\0')
    {
     point.type = PT_TYPE_INT32;
     break;
    }

    /* it's a double if conversion goes to end of string */
    eos = 0;
    point.value.r = strtod(ptvalue, &eos);
    if(*eos == '\0')
    {
     point.type = PT_TYPE_REAL;
     break;
    }

    /* else it's a string */

    point.type = PT_TYPE_STRING;
    point.value.s = ptvalue;
    break;
   }
 }

 /*
  * Write the point.  We need a IP_Msg structure and a IP_hTASK in
  * to provide buffer space and sender identification respectively.
  */
 if ((status = DH_WritePoint (htask, &point, hmsg, NULL)) != ST_OK)
  printf ("Write point failed: %s\n", ST_StatusName (status));

 return (0);
```

```
}
```

5.7.3. Registering for exceptions from the Cascade DataHub

```
/*
 * Cascade DataHub point waiter:  waiter.c
 *
 * (C) Copyright Cogent Real-Time Systems Inc., 1997.  All rights reserved.
 *
 * This program registers for exceptions on all or selected points
 * in the Cascade DataHub.
 *
 * The program waits in an infinite receive loop for exceptions.
 * Upon receipt of a message the program checks that the message is
 * a Casacde DataHub exception and then parses and prints the point
 * to stdout.
 *
 * Since this program can be made to register for exceptions on multiple
 * points a linked-list facility is used to create a list of point names
 * from the passed args.  This linked list is then walked to register
 * for exceptions on each point individually.
 *
 * This program is supplied with the Cascade DataHub programming API.  It
 * may be copied or modified, in whole or in part, for the sole purpose of
 * creating applications to be used with the Cascade DataHub.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>

#include <cogent.h>

#ifdef HAVE_SYS_KERNEL_H
#include <sys/kernel.h>
#endif

/*
 * millisecs function converts nanoseconds (10E-9) to milliseconds (10E-3).
 */
static int millisecs(PT_pCPOINT ppoint)
{
 return (ppoint->nanoseconds / 1000000);
}

/*
 * Generate an ASCII representation of the date and time for a given point,
 * truncating the nanoseconds to the next lowest millisecond.  If the time
 * values are 0,0, then generate the string "none".
 */

static char* timeof(PT_pCPOINT ppoint)
{
 static char ctm[64], msec[16];
 char  *thetime;
 time_t  tm;

 if (ppoint->seconds)
 {
  tm = ppoint->seconds;
  thetime = ctime (&tm);
  strncpy (ctm, thetime, 11);
  strncpy (&ctm[11], &thetime[20], 4);
  strncpy (&ctm[15], &thetime[10], 9);
  ctm[24]='\0';
  sprintf (msec, ".%03d", millisecs(ppoint));
```

```
  strcat (ctm, msec);
 }
 else
 {
  strcpy (ctm, "none");
 }
 return (ctm);
}

/*
 * Print point name, value and information to the standard output
 */
void print_point (PT_pCPOINT ppoint)
{
 printf ("Point: %s\n", ppoint->name);
 switch (ppoint->type)
 {
   case PT_TYPE_INT32:
  printf ("Value: %ld\n", (long)ppoint->value.i);
  break;
   case PT_TYPE_REAL:
  printf ("Value: %.20g\n", ppoint->value.r);
  break;
   case PT_TYPE_STRING:
  printf ("Value: %s\n", ppoint->value.s);
  break;
   default:
  printf ("Value: Unknown\n");
  break;
 }
 printf ("Conf:  %d, Lock: %s, Time: %s, Security: %d\n",
   ppoint->conf, (ppoint->locked ? "yes" : "no"),  timeof(ppoint),
   ppoint->security);
}

#ifdef __USAGE
%C [-d domain] [-q queuename] pointname...
 -d domain - set the domain for the operation
 -q        - name of queue for point changes
 -v        - print debugging information

 Points in another domain may be watched by
 texplicitly naming the domain followed by a colon
 and the point name.
  e.g.,
   waiter mixer:Mixer_1_Weight
  is the same as
         waiter -d mixer Mixer_1_Weight
#endif

const char* UT_USAGE =
 "Usage: %s [-d domain] [-q queuename] pointname...\n"
 ;

const char* UT_HELP =
    "\n"
    "Help:\n"
    "    -h          Print this helpful message.\n"
    "    -d domain   Set the domain for the operation.\n"
    "    -q          Name of queue for point changes.\n"
    "    -v          Print debugging information.\n"
    "\n"
    "   Points in another domain may be watched by\n"
    "   texplicitly naming the domain followed by a colon\n"
    "   and the point name.\n"
    "       e.g.,\n"
    "           waiter mixer:Mixer_1_Weight\n"
    "       is the same as\n"
```

```
    "           waiter -d mixer Mixer_1_Weight\n"
 ;

int main (int argc, char** argv)
{
 IP_Msg   *hmsg;
 ST_STATUS  status;
 PT_stCPOINT  point;
 char   *ptname = NULL, *qname = NULL, *domain=NULL;
 char   *s, namebuf[256], *name = NULL, *msgend, *myname;
 int    opt;
 int    debugging=0;
 IP_Task   *htask=NULL;
 IP_MsgInfo  msginfo;
 int    type;
 LL_LIST   names;
 LL_stITERATOR it;

 hmsg = IP_MsgCreate (NULL, IP_MsgDefaultSize(), 0);

 /*
   * Create an empty link list
  */
 names = LL_New();

 while ((opt = getopt(argc, argv, "hd:q:v")) != -1)
 {
  switch (opt)
  {
    case  'h':
     UT_Help(argv[0], UT_USAGE, UT_HELP);
   exit(0);
    case 'd':
   domain = optarg;
   if (strlen(domain) > 15)
    domain[15] = '\0';
   break;
    case 'q':
   qname = optarg;
   break;
    case 'v':
   debugging = 1;
   break;
    default:
   UT_Usage (argv[0], UT_USAGE, stderr);
   exit(1);
  }
 }
 for(;optind < argc; optind++)
 {
  /* Adds the arg to the end of the linked list */
  LL_AddTail (names, argv[optind]);
 }

 /*
  * If a queue name is not specified then generate one based on the
  * process pid
  */
 if (!qname)
 {
  sprintf (namebuf, "sc/wait%d", getpid());
  qname = strdup (namebuf);
 }
 if ((s = strrchr (argv[0], '/')))
  s++;
 else
  s = argv[0];
```

```
/*
 * Generate a registered name based on the process pid
 */
if (!name)
{
 sprintf (namebuf, "sc/wait%d", getpid());
 name = strdup (namebuf);
}

/*
 * Initialize this process with the name server and queue server
 */
if ((myname = strrchr(argv[0], '/')))
 myname++;
else
 myname = argv[0];

if (!(htask = IP_NserveInit (myname, domain, qname, 0, 0)))
{
 printf("IP_NserveInit() failed: are qserve and nserve running?\n");
 exit(1);
}

memset (&point, 0, sizeof(point));

/*
 * Traverse the link list of points (if passed) and register for
 * exceptions on each.  Otherwise register for all exceptions.
 */
if (LL_Count(names))
{
 LL_TRAVERSE (names, char*, ptname, it)
 {
  printf ("Register %s\n", ptname);
  point.name = ptname;
  point.conf = 0;
  point.address = NULL;
  if ((status = DH_RegisterPoint (htask, &point, hmsg, NULL))
   != ST_OK)
   printf ("Register point failed: %d\n", status);
 }
}
else
{
 if ((status = DH_RegisterAllPoints (htask, NULL, 1,
          hmsg, NULL)) != ST_OK)
 {
  printf ("Register all points failed: %d\n", status);
  exit (-1);
 }
}

/*
 * Provide a point name buffer separately from the rest of the point
 * structure.  There is no way for the API to know what the allocation
 * status of a point name is, so it will never attempt to free this
 * buffer, nor write into it.
 */
point.name = namebuf;

/*
 * Infinite event loop
 */
for (;;)
{
 /*
  * Sit receive blocked
  */
```

```
type = IP_Receive (htask, hmsg, &msginfo);
if (debugging)
 printf ("Received: %s\n", (char*)IP_MsgData (hmsg));

/*
 * If the message is an exception notice from the Cascade DataHub
 * then parse and print the point.  We could get messages from other
 * sources as well, which we effectively ignore.  If any process
 * attempts to Send a message to this task, we just send back a nil
 * response.
 */
switch (type)
{
  case IP_ASYNC:   /* DataHub point, probably */
 switch (IP_MsgSubtype (hmsg))
 {
   case ST_DH_EXCEPTION:
  /* Deal with exceptions by parsing and printing the result */
  for (msgend = IP_MsgData(hmsg); *msgend; )
  {
   DH_ParsePointString (&point, point.name, msgend,
         &msgend, NULL);
   print_point (&point);
  }
  break;
   case ST_DH_ECHO:
  /* We never write a point, so ECHO is impossible in this
   * app.  If we expected echoes, we could handle them in
   * the same way as exceptions.  The message contents are
   * identical for both. */
  break;
   default:
  /* Async general ASCII message, sent by another task using
   * the Cascade DataHub API.  One such task is the Cascade
   * name server, nserve.  It will send taskstarted and
   * taskdied messages when other tasks start and stop:
   * (taskstarted <name> <domain> <queue> <node> <pid> <chid>)
   * (taskdied <name> <domain> <queue> <node> <pid> <chid>)
   */
  break;
 }
 break;
  case IP_ERROR:
 /* Receive returned zero, which should never happen. */
 break;
  case IP_SYSTEM:
 /* A task death message that can only be received by a task
  * that has set its QNX INFORMED bit.  You should avoid this,
  * and use the taskdied and taskstarted messages in the
  * IP_ASYNC section if possible. */
 break;
  case IP_SIGNAL:
 /* A signal caused the Receive to exit prematurely.  No message
  * was received.  The value of sender is not defined. */
 break;
  case IP_NONE:
 /* The queue reported a message waiting, but none was available.
  * This happens if the application intentionally drains the queue,
  * which is generally a bad idea. */
 break;
  case IP_SYNC:
 /* Synchronous general ASCII message, sent by another task using
  * the Cascade DataHub API. */
 IP_MsgCascade (hmsg, "Unsupported", 12, IP_NONE, ST_ERROR);
 IP_MsgInfoReply (&msginfo, hmsg);
 break;
  case IP_RAW:
 /* Message sent using QNX Send command, not through the
```

```
    * Cascade DataHub API.  This is also where a proxy would
    * be delivered if we had set up a proxy within this task,
    * for example, a periodic timer.  In that case, the proxy
    * ID would be (pid_t)sender, and we would not Reply to it. */
   IP_MsgInfoReplyRaw (&msginfo, "Unsupported", 12);
   break;
  }
 }

 return 0;
}
```

5.7.4. A sample makefile definition

```
    COGLIB = -l cogdb
    CFLAGS = -Oneatx -Q

    all: readpt writept waiter

    readpt: readpt.c
            cc -o $@ $< $(COGLIB)

    writept: writept.c
            cc -o $@ $< $(COGLIB)

    waiter: waiter.c
            cc -o $@ $< $(COGLIB)

    clean:
            rm -f *.o readpt writept waiter

    .PHONY: clean
    .IGNORE: clean
```

Chapter 6. The Cascade Historian

The Cogent C API for the Cascade Historian consists of C functions that are wrappers around the Cascade Historian command set. The C syntax for these functions is documented in the Cascade Historian Functions reference section of this manual, while the command syntax is documented in the Cascade Historian manual. Please refer to that document for general information on the Cascade Historian.

6.1. Command/Function Correspondence

The following table shows all the commands and functions available in the Cascade Historian and the corresponding Cogent C API functions, illustrating how they correspond to each other. This table is also available in the Command/Function Correspondence section of the Cascade Historian manual.

Table 6-1. Cascade Historian Commands and Functions

Command	API Function	DLL Function
add	HI_Add	-
apropos	-	-
aproposSyntax	-	-
bufferIdData	HI_BufferIDRead	hist_buffer_id_read
bufferIdDataAscii	-	-
bufferIdDestroy	HI_BufferIDDestroy	hist_buffer_id_destroy
bufferIdLength	HI_BufferIDLength	hist_buffer_id_length
bufsize	HI_Bufsize	hist_bufsize
count	HI_Count	hist_count
deadband	HI_Deadband	hist_deadband
delete	HI_Delete	hist_delete
describe	HI_Describe	hist_describe
disable	HI_Disable	hist_disable
earliest	HI_Earliest	hist_earliest
enable	HI_Enable	hist_enable
exit	-	-
flag	-	-
filebase	HI_FileBase	hist_filebase
flush	HI_Flush	hist_flush
histdb	-	-
history	HI_History	hist_history
include	-	-
interpolate	HI_Interpolate	hist_interpolate
-	HI_InterpolateData	hist_interpolate_data
interpolatorDescribe	-	-
interpolatorList	-	-
latest	HI_Latest	hist_latest
length	HI_Length	hist_length

Command	API Function	DLL Function
list	HI_List	hist_list
register	HI_Register	hist_register
unregister	HI_Unregister	hist_unregister
version	HI_Version	hist_version

6.2. Binary Data Buffer Functions

The following table lists a set of functions for accessing and performing some commonly needed transformations on the binary data buffers in the Cascade Historian. This table is also available in the Binary Data Buffer Functions section of the Cascade Historian manual.

Table 6-2. Cascade Historian Binary Data Buffer Functions

API Function	DLL Function
-	hist_access_buffer
-	hist_buffer2array
HI_ClipBuffer	hist_clip_buffer
HI_ExchangeBuffer	hist_exchange_buffer
HI_GapCountBuffer HI_GapFillBuffer	hist_gap_buffer
-	hist_length_buffer
HI_ScaleBuffer	hist_scale_buffer
HI_StatBuffer	hist_stat_buffer

Chapter 7. Cogent Driver Specifications

The Cogent Driver API functions can be grouped according to their purpose, as shown below. All these functions are described in detail in the Reference. They are intended to be quite generic, to provide a consistent interface for different types of cards. However each specific driver has its own protocol-specific parameters, which are detailed in this chapter.

7.1. Cogent Driver Functions

Connection and Command Functions

`DR_ApInitIPC`
`DR_ApConnectIPC`
`DR_ApCloseIPC`
`DR_ApCommand`

Point Interface

`DR_ApReadPoint`
`DR_ApWritePoint`
`DR_ApListPoints`
`DR_ApDescribePnt`
`DR_ApPointBufAddress`

Block Interface

The number, type and length of the blocks accessed by this interface depends on the specific driver.

`DR_ApReadBlock`
`DR_ApWriteBlock`
`DR_ApListBuffers`
`DR_ApDescribeBuffer`
`DR_ApUpdateBuffers`
`DR_ApReadStatus`
`DR_ApReadControl`
`DR_ApWriteControl`

7.2. Hilscher Fieldbus CIF Card

For more information on the CIF Driver, please refer to the Cogent CIF Driver for Hilscher Fieldbus CIF Cards manual.

7.2.1. I/O Block Functions

The `DR_ApReadBlock` and `DR_ApWriteBlock` functions access the process I/O data area of the card. The offset and size parameters specify what portion of the buffer area to transfer, and is dependent on the field configuration. Typically, field data is always mapped to the beginning of the block (offset of `0`) and is continguous up to the number of bytes defined by the field device (slave) configuration. The output process data buffer is mapped as buffer `0`, while `1` will access the input process data buffer.

7.2.2. Control Block Functions

The `DR_ApReadControl` and `DR_ApWriteControl` functions access the control parameter blocks of the card. Only buffer 2 is valid for CIF cards, which provides access to the protocol parameters.

The control parameters for the Hilscher Fieldbus card use the following structure (see file `cif_api.h`):

```
typedef struct {
     unsigned char Mode;
```

```
        unsigned char reserved1;
        unsigned char Format;
        unsigned short WatchdogTime;
        unsigned char reserved5[3];
        unsigned char reserved8[8];
    } cif_ApParms_t;
```

The control parameters do vary with the specific fieldbus protocol and additional structure definitions can be found in the `cif_api.h` file.

Access to the control data is accomplished via buffer 2 with a length of 16, using the functions `DR_ApReadControl` and `DR_ApWriteControl`, as follows:

```
DR_ApReadControl (dev, 2, 0, 16, &control, &error)
DR_ApWriteControl (dev, 2, 0, 16, &control, &error)
```

It is recommended that reserved bytes be set to `0`. Most protocols reserve the last 11 bytes, so the length may be shortened to `5`, depending on the protocol. Note that a read of the control parameters may return only zero data from some cards.

The components of the control structure are defined as follows:

Parameter	Description	Length
Mode	Type of handshake mechanism for process data delivery.	byte
Cycle_time	Cycle time of the fieldbus cycle (hwere applicable).	byte
Format	Storage format of word data.	byte
WatchdogTime	HOST-supervision time in multiples of a msec.	short

The handshake modes available are:

Mode	Name (see cif_api.h)	Description
0	IOMODE_DEV_UNBUF	card controlled, bus synchronous data transfer
1	IOMODE_DEV_BUF	card controlled, buffered data transfer
2	IOMODE_NO_CTL	no handshake
3	IOMODE_HOST_BUF	HOST controlled, buffered data transfer
4	IOMODE_HOST_UNBUF	HOST controlled, bus synchronous data transfer

Not all modes are available for all protocols.

The available storage formats are:

Format	Name (see cif_api.h)	Description
0	FORMAT_INTEL	Intel; little-endian; low-byte, high-byte
1	FORMAT_MOTORLA	Motorola; big-endian; high-byte, low-byte

After writing the control block, the system must do a WARM reset before the changes will take effect (see the **cardReset** command in the Device Driver for Hilscher CIF Cards manual).

7.2.3. Status Block Functions

The `DR_ApReadStatus` function accesses the status blocks of the card. Only buffer `2` is valid for CIF cards, which provides master status as well as field device (slave) status and diagnostic information.

The status of the Hilscher Fieldbus Card is read from buffer 2 with a length of 64, using the function `DR_ApReadStatus`, as follows:

```
DR_ApReadStatus (dev, 2, 0, 64, &status, &error)
```

The offset and length are ignored. Status is a structure of type cif_ApState_t (see the file `cif_api.h`) and is defined as follows:

```
typedef struct {
    unsigned char global_bits;
    unsigned char bus_status;
    unsigned char err_rem_addr;
    unsigned char err_rem_event;
    unsigned char reserved[28];
    unsigned char state [16];
    unsigned char diag [16];
} cif_ApState_t;
```

where:

State Element	Description	Length
`global_bits`	Error bits, as follows: bit 0: Ctrl, parameterization error bit 1: Aclr, slave error causing AutoClear mode bit 2: Ndata: at least one slave not in data exchange mode or reporting an error bit 3-7: reserved	byte

`bus_status`	Main state of the master system:
	`0x00 OFFLINE`
	`0x40 STOP`
	`0x80 CLEAR`
	`0xC0 OPERATE`
		byte

`err_rem_addr`	Remote address of error source	byte
`err_rem_event`	Error number	byte
`state`	A bitfield classifying every slave as active (1) or inactive (0)	16 x byte
`diag`	A bitfield showing diagnostic bit of every slave	16 x byte

Appendix A. GNU General Public License

GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program",

below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

Section 1

You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

Section 3

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or

otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE

PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type “show w”. This is free software, and you are welcome to redistribute it under certain conditions; type “show c” for details.

The hypothetical commands “show w” and “show c” should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than “show w” and “show c”; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program “Gnomovision” (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix B. GNU Lesser General Public License

GNU Lesser General Public License

Version 2.1, February 1999

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method:

1. we copyright the library, and
2. we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author’s reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the *Lesser* General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

Section 0

This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

Section 1

You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. The modified work must itself be a software library.

b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

   (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

Section 3

You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

Section 4

You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

Section 5

A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a “work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a “work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a “work that uses the library”. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a “work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

Section 6

As an exception to the Sections above, you may also combine or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work

during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable “work that uses the Library”, as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user’s computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the “work that uses the Library” must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

Section 7

You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

Section 8

You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who

have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 9

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

Section 10

Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

Section 11

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 12

If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 13

The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

Section 14

If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY Section 15

BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Section 16

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the library’s name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library ’Frob’ (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990 Ty Coon, President of Vice

That’s all there is to it!

II. Reference

Table of Contents

I. Utilities 50
II. Data Types 58
III. Cascade DataHub Functions 63
IV. Cogent Driver Functions 88
V. Cascade Historian Functions 117
VI. Inter-Process Communication Functions 159
VII. Cascade TextLogger Functions 249
VIII. Point Manipulation Functions 272

I. Utilities

Table of Contents

lsend, gsend **51**
nsnames **53**
nserve **55**
qserve **57**

These utilities are commonly available to Cogent software.

lsend, gsend

lsend, gsend — send commands to Cogent and Gamma programs.

Synopsis

lsend [-ghlVX][*task*]
gsend [-ghlVX]*task*

Arguments

task

A Cogent or Gamma program name (as a string, previously attached by name_attach, init_ipc, or qnx_name_attach), or a process ID (which is O/S dependent).

-g

Accept Gamma input. This is generally used from **lsend**, and is the equivalent of running **gsend**. It is the default for **gsend**.

-h

Print a help message and exit.

-l

Accept Lisp input. This is generally used from **gsend**, and is the equivalent of running **lsend**. It is the default for **lsend**.

-V

Print the version number.

-X

Exit immediately (usually used with -V).

Returns

A prompt with the *task* name displayed.

Description

The **lsend** utility attaches to a running Gamma program and allows the user to send commands without exiting the event loop of the attached process. Any statement may be issued, including changing the definitions of existing functions. **lsend** statements use Lisp syntax, which consists of the name of the command or function, followed by a space-separated list of arguments, all enclosed in parentheses, like this:

```
(command arg1 arg2 arg3 ...)
(function arg1 arg2 arg3 ...)
```

The **gsend** utility is simply a symbolic link to **lsend**, and is the equivalent of calling **lsend** with the -g option. **gsend** statements use Gamma syntax, which is slightly different from Lisp syntax. The command or function name is outside the parentheses, and the arguments are separated by commas, like this:

```
command (arg1, arg2, arg3, ...)
function (arg1, arg2, arg3, ...)
```

To exit **lsend** or **gsend**, use **Ctrl-C** or **Ctl-D**. Anything else you type will be parsed and passed on to the task you have been communicating with.

Event processing stops in the Gamma program for the duration of either of these commands.

Example

Sending a Lisp command to an example Gamma process named `OtherProcess`:

```
[sh]$ lsend "OtherProcess"
OtherProcess> (cos 5.5)
0.70866977429126
OtherProcess>
```

Sending a Gamma command to the same process:

```
[sh]$ gsend "OtherProcess"
OtherProcess>cos (5.5)
0.70866977429126
OtherProcess>
```

The example process:

```
Gamma> init_ipc ("OtherProcess");
t
Gamma> while(t) next_event();
```

It is possible to send several commands at once, by typing them one after another. Here is an example using **lsend** with the Cascade Historian, running under the task name `Hist`:

```
[sh]$  lsend Hist
Hist> (disable p*)(enable ptbogus)(history p6)
t
Hist> (error "histdb: enable: No history found for: ptbogus")
Hist> t
Hist>
```

lsend sends commands and shows the return values until all the commands have been attempted. Each command is processed separately by the Cascade Historian. **lsend** displays a prompt and the return value for each command separately.

See the Sending Commands section of the Cogent Tools Demo for another example of this command in use.

nsnames

`nsnames` — queries the Cascade NameServer.

Synopsis

nsnames `[-DhNQsSUVX][-d` *`domain`*`][-e` *`node`*`][-p` *`pid`*`][-q` *`queue`*`][-t` *`task`*`]`

Arguments

`-d` *`domain`*

Report only on domains matching this shell expression.

`-D`

Sort by domain.

`-e` *`node`*

Report only on this node.

`-h`

Print this helpful message and exit.

`-N`

Sort by node.

`-p` *`pid`*

Report only on this process ID.

`-q` *`queue`*

Report only on queues matching this shell expression.

`-Q`

Sort by queue name.

`-s`

Supress headers.

`-S`

Sort by name (default).

`-t` *`task`*

Report only on tasks matching this shell expression.

`-U`

Unsorted output.

`-V`

Print the version number.

`-X`

Exit immediately (normally used with `-V`).

Returns

The specified information, listed to the console.

Description

This module retrieves name information from **nserve**, according to optional criteria. A *`pid`* or *`node`* of `0` matches any process ID or node ID. The *`domain`*, *`qname`*, and *`task`* match on substrings, so "`/cogent/ll`" and "`/cogent/hl`" would both be matched by: `-t cogent`.

Dependencies

nserve

See Also

Using the Cascade DataHub, nserve

nserve

`nserve` — starts the Cascade NameServer.

Syntax

nserve `[-DhirvVX][-p` *`freqncy`*`][-s` *`freqncy`*`]`

Arguments

`-D`

Do not detach from the console. Run in the foreground.

`-h`

Print a help message and exit.

`-i`

Run in isolation. Refuse tasks from other nodes. If `-r` is not specified, other nodes will still see tasks on this node.

`-p` *`freqncy`*

Poll for dead network nodes every *`freqncy`* seconds (default is 5).

`-r`

Do not run a network redirector. No other name server on the network will see tasks on this node. This name server will still see tasks on other nodes.

`-s` *`freqncy`*

Check for dead name servers on live nodes every *`freqncy`* polls (default is 1).

`-v`

Generate debugging output. Implies `-D`.

`-V`

Print version number.

`-X`

Exit immediately (normally used with `-V`).

Returns

On success, nothing; on error, a message.

Description

This module provides a network-wide name service for Cogent tasks. These tasks include, but are not limited to, Cascade DataHub, Cascade Historian, Cascade TextLogger, SCADALisp, and Gamma. This module is started automatically by them, after the **qserve** module.

This module should be run before the TCP/IP **Socket** command is run. If **Socket** is started before **nserve** then a race condition may occur where a MS-Windows node connects via TCP but sees no **nserve** module running and incorrectly assumes that it is running without a Cascade DataHub. Normally **qserve** and **nserve** are placed in the `sysnit.N` file of the server. Names give tasks a method of identification on one or many PCs. The **nserve** task is an active name server, and (unlike QNX's passive **nameloc** module) notifies all its tasks of start and stop events which occur in other registered tasks.

The **nserve** program also allows access to other registered tasks' queues.

Dependencies

qserve

See also

Cascade NameServer, Using the Cascade DataHub, nsnames

qserve

`qserve` — starts the Cascade QueueServer.

Synopsis

qserve `[-DhvVX]`

Arguments

`-D`

Do not detach from the console. Run in the foreground.

`-h`

Print a help message and exit.

`-v`

Generate debugging information. (Implies the use of `-D`).

`-V`

Print the version number.

`-X`

Exit immediately (usually used with `-V`).

Returns

On success, nothing; on error, a message.

Description

The **qserve** utility is an asynchronous queue manager for Cogent applications such as the Cascade DataHub, Cascade Historian, Cascade TextLogger, and Gamma. The **qserve** utility is started automatically by any of these programs when they start up. It must be the first IPC-related task to be started, followed by **nserve**.

It is possible for a task to have the same queue open for both read and write (only one open of each type) without interfering with one another.

Dependencies

none

See Also

Using the Cascade DataHub, datahub

II. Data Types

Table of Contents

HI_stVALUE 59
PT_stCPOINT 60
PT_TYPE, PT_uVALUE 61
ST_STATUS 62

These are data types and structures commonly referenced in the Cascade DataHub API.

HI_stVALUE

HI_stVALUE — contains an (x,y) data tuple used by the Cascade Historian.

Synopsis

```
typedef struct HI_stVALUE
{
    double xaxis;
    double value;
} HI_stVALUE;
```

Members

xaxis	X data value of the tuple, typically time expressed in seconds.
value	Y data value of the tuple, typically the historical data of interest.

Description

This simple structure provides a generalized (x,y) data pair from which histories and queries are built. A history, being a time-series of data, uses time as the x axis and therefore interprets the structure as (time, value). Queries may also provide data vs. time, but may also provide data against some other variable.

This structure is analogous to the Cascade Historian class HI_stVALUE.

PT_stCPOINT

PT_stCPOINT — holds information about a Cascade DataHub point.

Synopsis

```
typedef struct PT_stCPOINT
{
short type;
PT_uVALUE value;
short conf;
char *name;
short locked;
short security;
void *address;
int32 seconds;
int32 nanoseconds;
void *userdata;
} PT_stCPOINT;

typedef PT_stCPOINT *PT_pCPOINT;
```

Members

At least the following fields are defined:

`type`

The type of the DataHub point's data. This can be one of:

`PT_TYPE_INT32`

An integer.

`PT_TYPE_REAL`

A floating point number.

`PT_TYPE_STRING`

A character string.

Description

Holds information about a point in the Cascade DataHub.

See Also

Point Structure, Storage, and Manipulation

PT_TYPE, PT_uVALUE

PT_TYPE, PT_uVALUE — holds the type and value a Cascade DataHub point.

Synopsis

```
/* A point type can be at least one of the following: */
enum PT_TYPE
{
PT_TYPE_STRING,
PT_TYPE_REAL,
PT_TYPE_INT32,
};

/* A points value is stored in the following union: */
typedef union
  {
  ptreal r;
  int32 i;
  char *s;
  } PT_uVALUE;

  typedef PT_uVALUE *PT_pVALUE;
```

Members

At least the following fields are defined:

r

Either a single-precision or a double-precision floating point number.

i

A 32-bit signed integer regardless of the machine architecture or compiler.

s

A C-style (null-terminated) string of characters.

See Also

Point Structure, Storage, and Manipulation

ST_STATUS

ST_STATUS — contains return values for some Cogent C API functions.

Synopsis

```
typedef enum
{
    ST_OK = 0, ST_ERROR, ST_NO_TASK, ST_NO_MSG, ST_WOULDBLOCK, ST_INTR,
    ST_FULL, ST_LOCKED, ST_SECURITY, ST_NO_POINT, ST_INSIG, ST_UNKNOWN,
    ST_NO_QUEUE, ST_CMD_SYNTAX_ERROR, ST_REPLIED, ST_WRONG_TYPE,
    ST_TOO_LARGE, ST_NO_MEMORY, ST_OLD_DATA, ST_TIMEOUT
} ST_STATUS;
```

Members

The following descriptions are general, and should be interpreted in the context of the specific function and circumstances.

ST_OK	The function executed without error.
ST_ERROR	An error occurred.
ST_NO_TASK	A required task does not exist.
ST_NO_MSG	There is no message available.
ST_WOULDBLOCK	This action would block, and is not permitted.
ST_INTR	An interrupt occurred.
ST_FULL	The queue is full.
ST_LOCKED	A Cascade DataHub point is locked.
ST_SECURITY	The security level is insufficient.
ST_NO_POINT	A required Cascade DataHub point does not exist.
ST_INSIG	A change in a Cascade DataHub point's value is insignificant. This is not really an error, but a notification that no exception will be generated by the datahub.
ST_UNKNOWN	There is an unknown error.
ST_NO_QUEUE	A target task has no queue, or **qserve** is absent.
ST_CMD_SYNTAX_ERROR	The command was not found, or there was a syntax error.
ST_REPLIED	The reply was complete.
ST_WRONG_TYPE	The type of a point or variable was wrong.
ST_TOO_LARGE	A value to be written to memory is larger than the available buffer.
ST_NO_MEMORY	There is insufficient memory available.
ST_OLD_DATA	Time-significant data is out of date.
ST_TIMEOUT	A timeout occurred in poll mode.

Description

This structure contains return values for some Cogent C API functions.

III. Cascade DataHub Functions

Table of Contents

DH_AppendString 64
DH_CreatePoint 65
DH_FindPointAddress 66
DH_FormatPoint 67
DH_ParsePointMsg 69
DH_ParsePointString 71
DH_PointAdd, DH_PointDivide, DH_PointMultiply 73
DH_ReadPoint, DH_ReadExistingPoint 75
DH_RegisterAllPoints 77
DH_RegisterPoint, DH_RegisterExistingPoint 78
DH_SendPointMessage 80
DH_SetLock, DH_SetSecurity 81
DH_SetReceiveFormat 82
DH_SetTransmitFormat 84
DH_UnregisterPoint 85
DH_WritePoint, DH_WriteExistingPoint, DH_WriteExistingPoints 86

DH_AppendString

DH_AppendString — appends a string to a point.

Syntax

```
#include <cogent.h>
ST_STATUS DH_AppendString (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   char* str,
   IP_Msg* hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

ppoint

A pointer to a Cascade DataHub point structure. All fields are valid, and will be treated by the write, but in the first call on a point the address field must be zero. It will be filled in automatically. This address will be valid so long as the point exists in the DataHub, and if, in subsequent DH_* functions, the address is provided for this point, the function will run slightly faster.

str

The string of characters to be appended to the existing point value.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function appends a string to the value of a point. It is an atomic function that operates in one step, without reading or evaluating the existing value of the point. This provides a way to modify the value quickly, without interference or serialization problems.

See Also

DH_PointAdd

DH_CreatePoint

DH_CreatePoint — constructs a point.

Syntax

```
#include <cogent.h>
ST_STATUS DH_CreatePoint (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   IP_Msg* hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

ppoint

A pointer to a Cascade DataHub point structure. All fields are valid, and will be treated by the write, but in the first call on a point the address field must be zero. It will be filled in automatically. This address will be valid so long as the point exists in the DataHub, and if, in subsequent DH_* functions, the address is provided for this point, the function will run slightly faster.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function constructs a point in the Cascade DataHub. The value and confidence assigned to the point are both 0.

See Also

DH_ReadPoint, DH_RegisterPoint, DH_WritePoint

DH_FindPointAddress

DH_FindPointAddress — is for internal use only.

Syntax

```
#include <cogent.h>
ST_STATUS DH_FindPointAddress (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   ER_hLIST elist
);
```

Description

This function is for internal use only.

DH_FormatPoint

DH_FormatPoint — puts point data in ASCII format.

Syntax

```
#include <cogent.h>
int DH_FormatPoint (
   char* command,
   IP_Task* myself,
   PT_pCPOINT ppoint,
   char* buf,
   int len,
   int flags
);
```

Arguments

command

The first word in the command string. It can't be NULL, but it can be blank.

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

ppoint

A pointer to a Cascade DataHub point structure, which must contain at least name. All other fields in the point structure should be 0 if this is the first call on this point, because they will be filled in by this call. Subsequent calls on the same point do not require zeroing any fields. If one point structure is used with several point names, the address field must be zeroed each time a different name is used in a call to this function.

buf

The buffer to be written into.

len

The length of *buf*.

flags

One or none of these:

PT_FMT_ASCII	Constructs a humanly readable ASCII string (the default).
PT_FMT_BINARY	Encodes data as a raw binary representation, efficient for parsing.
PT_FMT_HEX	Encodes data as ASCII encoded hex. (This is not a valuable format since it is more difficult to parse and less space efficient than binary, and it is more difficult for humans to read than regular ASCII.)
PT_FMT_POINT_SECURITY	Takes a security value from the point structure, not a task structure.

Returns

The length of the buffer.

Description

This function takes a point structure and creates an ASCII representation of data for the point in the *buf* buffer. The data is represented as a list, in this format:

```
(command pointname type value conf security locked seconds nanoseconds)
```

The resulting buffer can be passed back to `DH_ParsePointString`, which would parse it back out. It infers the flag.

See Also

PT_stCPOINT, Point Structure, Storage, and Manipulation

DH_ParsePointMsg

DH_ParsePointMsg — parses a point message from the Cascade DataHub.

Syntax

```
#include <cogent.h>
ST_STATUS DH_ParsePointMsg (
   PT_pCPOINT ppoint,
   char* name,
   IP_hMSG hmsg,
   ER_hLIST elist
);
```

Arguments

ppoint

A pointer to a point structure which will be filled in by this function. The name element of the structure is not filled in, but is placed in the *name* argument instead.

All other fields are valid, and will be treated by the write, but in the first call on a point the address field must be zero. It will be filled in automatically. This address will be valid so long as the point exists in the DataHub, and if, in subsequent DH_* functions, the address is provided for this point, the function will run slightly faster.

name

A pointer to a buffer which will be filled in with the point's name by this function.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS. ST_ERROR means the buffer did not contain a valid point specification.

Description

This function parses the return value from DH_ReadPoint, DH_RegisterPoint, or a message received through an asynchronous message (exception) from the DataHub, and places the result in the provided point structure.

The name of the point is not modified in the point structure, as there is no clear manner in which the name should be handled. This leaves it up to the programmer to perform the memory allocation and deallocation of the point name storage as required. This function will only treat a single point specification in the message, ignoring all points after the first.

See Also

Communicating with the Cascade DataHub, DH_ParsePointString

DH_ParsePointString

DH_ParsePointString — parses the return of DH_ReadPoint and DH_RegisterPoint.

Syntax

```
#include <cogent.h>
ST_STATUS DH_ParsePointString (
   PT_pCPOINT ppoint,
   char* name,
   char* msgdata,
   char** msgend,
   ER_hLIST elist
);
```

Arguments

ppoint

A pointer to a point structure which will be filled in by this function. The name element of the structure is not filled in, but is placed in the *name* argument instead.

All other fields are valid, and will be treated by the write, but in the first call on a point the address field must be zero. It will be filled in automatically. This address will be valid so long as the point exists in the DataHub, and if, in subsequent DH_* functions, the address is provided for this point, the function will run slightly faster.

name

A pointer to a buffer which will be filled in with the point name by this function.

msgdata

A pointer to a character string which contains one or more point specifications. This can be the buffer portion of an IP_hMSG, available using the IP_MsgBuffer function.

msgend

Returns a pointer to a (char*) variable which will point to the character following the end of the point specification parsed by this call. This pointer may be passed back to a subsequent call to DH_ParsePointString to handle more than one point specification in a single incoming message, as may occur for a point exception.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS. ST_ERROR means the *msgdata* did not contain a valid point specification.

Description

This function parses the return value from DH_ReadPoint and DH_RegisterPoint, or a message received through an asynchronous message (exception) from the DataHub, and places the result in the provided point structure.

The name of the point is not modified in the point structure, as there is no clear manner in which the name should be handled. This leaves it up to the programmer to perform the memory allocation and deallocation of the point name storage as required. The *msgend* argument allows the programmer to deal with more than one point definition in a single message, which is typical of DataHub exception messages. When the last point definition in the message has been parsed, then *msgend* will point to a zero length string (`**msgend == '\0'`).

See Also

Communicating with the Cascade DataHub, DH_ParsePointMsg

DH_PointAdd, DH_PointDivide, DH_PointMultiply

DH_PointAdd, DH_PointDivide, DH_PointMultiply — modify a point in place, with one message.

Syntax

```
#include <cogent.h>
ST_STATUS DH_PointAdd (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   double value,
   IP_Msg* hmsg,
   ER_hLIST elist
);
ST_STATUS DH_PointDivide (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   double value,
   IP_Msg* hmsg,
   ER_hLIST elist
);
ST_STATUS DH_PointMultiply (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   double value,
   IP_Msg* hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

ppoint

A pointer to a Cascade DataHub point structure. All fields are valid, and will be treated by the write, but in the first call on a point the address field must be zero. It will be filled in automatically. This address will be valid so long as the point exists in the DataHub, and if, in subsequent DH_* functions, the address is provided for this point, the function will run slightly faster.

value

The value to be used to modify the value of the point.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

These functions give an atomic way to modify a value in the DataHub in place, using a single message. Normally a value is modified in three steps: read, modify, and write. This functions offers a way to modify a value without interference or serialization problems.

The operations provided are as follows:

`DH_PointAdd`	Adds the *`value`* to the point.
`DH_PointDivide`	Divides the point by the *`value`*.
`DH_PointMultiply`	Multiplies the point by the *`value`*.

See Also

DH_AppendString

DH_ReadPoint, DH_ReadExistingPoint

DH_ReadPoint, DH_ReadExistingPoint — read a point from the Cascade DataHub.

Syntax

```
#include <cogent.h>
ST_STATUS DH_ReadPoint (
   IP_hTASK myself,
   PT_pCPOINT ppoint,
   IP_hMSG hmsg,
   ER_hLIST elist
);
ST_STATUS DH_ReadExistingPoint (
   IP_hTASK myself,
   PT_pCPOINT ppoint,
   IP_hMSG hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

ppoint

A pointer to a Cascade DataHub point structure, which must contain at least name. All other fields in the point structure should be 0 if this is the first call on this point, because they will be filled in by this call. Subsequent calls on the same point do not require zeroing any fields. If one point structure is used with several point names, the address field must be zeroed each time a different name is used in a call to this function.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

inter

Process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function reads a point from the DataHub. If the named point does not exist then DH_ReadPoint will create it on the DataHub with a confidence factor of zero and DH_ReadExistingPoint will return the error ST_NO_POINT.

See Also

Communicating with the Cascade DataHub, DH_CreatePoint, DH_WritePoint, DH_RegisterPoint

DH_RegisterAllPoints

DH_RegisterAllPoints — registers with the Cascade DataHub for all points.

Syntax

```
#include <cogent.h>
ST_STATUS DH_RegisterAllPoints (
   IP_hTASK myself,
   char* domain,
   int future,
   IP_hMSG hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

domain

The DataHub domain on which to register. If the domain is NULL, then the current domain is used and all exceptions will be transmitted with unqualified names. If a valid domain is specified, then all exceptions will be transmitted with qualified names, even if the domain is the current domain for the application.

future

A flag indicating whether points which are created on the DataHub subsequent to this call should also be registered automatically. 0 = do not automatically register future points. 1 = automatically register future points

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function informs the DataHub that this task should receive exceptions on all points. If the future flag is set, then the DataHub will automatically begin to inform the task of any points which are created on the DataHub subsequent to this call. This call will cause an exception to be generated from the DataHub for every point, specifically to this task (not to other tasks that may also be registered).

See Also

Communicating with the Cascade DataHub, DH_RegisterPoint

DH_RegisterPoint, DH_RegisterExistingPoint

DH_RegisterPoint, DH_RegisterExistingPoint — register with the Cascade DataHub for a point.

Syntax

```
#include <cogent.h>
ST_STATUS DH_RegisterPoint (
   IP_hTASK myself,
   PT_pCPOINT ppoint,
   IP_hMSG hmsg,
   ER_hLIST elist
);
ST_STATUS DH_RegisterExistingPoint (
   IP_hTASK myself,
   PT_pCPOINT ppoint,
   IP_hMSG hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

ppoint

A pointer to a Cascade DataHub point structure, which must contain at least name. All other fields in the point structure should be 0 if this is the first call on this point, because they will be filled in by this call. Subsequent calls on the same point do not require zeroing any fields. If one point structure is used with several point names, the address field must be zeroed each time a different name is used in a call to this function.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function informs the DataHub that this task should receive exceptions (value change notifications) for the named point. The DataHub will not generate an initial exception on the point, but instead returns the current value of the point in the *ppoint* structure. Subsequent exceptions will be transmitted to the task as type=IP_ASYNC messages with command=ST_EXCEPTION or command=ST_ECHO. DH_RegisterPoint will create the point on the DataHub if it does not exist. DH_RegisterExistingPoint will return an error if the point does not exist.

See Also

Communicating with the Cascade DataHub, DH_UnregisterPoint DH_CreatePoint, DH_ReadPoint, DH_WritePoint, and DH_ParsePointMsg

DH_SendPointMessage

DH_SendPointMessage — is for internal use only.

Syntax

```
#include <cogent.h>
ST_STATUS DH_SendPointMessage (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   IP_Msg* hmsg,
   ER_hLIST elist
);
```

Description

This function is for internal use only.

DH_SetLock, DH_SetSecurity

DH_SetLock, DH_SetSecurity — set the lock or security according to point status.

Syntax

```
#include <cogent.h>
ST_STATUS DH_SetLock (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   IP_Msg* hmsg,
   ER_hLIST elist
);
ST_STATUS DH_SetSecurity (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   IP_Msg* hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

ppoint

A pointer to a Cascade DataHub point structure, which must contain at least name. All other fields in the point structure should be 0 if this is the first call on this point, because they will be filled in by this call. Subsequent calls on the same point do not require zeroing any fields. If one point structure is used with several point names, the address field must be zeroed each time a different name is used in a call to this function.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

These functions set or unset the Cascade DataHub lock or security for a point according to the status of the point itself. They look at the point structure, and set the lock or security status in the DataHub to be the same as the point's locked or security value.

DH_SetReceiveFormat

DH_SetReceiveFormat — sets DataHub-to-client transmissions to binary or ASCII.

Syntax

```
#include <cogent.h>
ST_STATUS DH_SetReceiveFormat (
   IP_Task* myself,
   IP_Msg* hmsg,
   char* domain,
   int flag,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

domain

The name of the Cascade DataHub domain.

flag

One of these:

PT_FMT_ASCII	Constructs a humanly readable ASCII string (the default).
PT_FMT_BINARY	Encodes data as a raw binary representation, efficient for parsing.
PT_FMT_HEX	Encodes data as ASCII encoded hex. This is not a valuable format since it is more difficult to parse and less space efficient than binary, and it is more difficult for humans to read than regular ASCII.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function sets a global variable that determines whether transmissions from the Cascade DataHub to a client will be in binary or ASCII format. It instructs the DataHub to transmit all messages to the custom client using the specified format. This call will fail with ST_NO_TASK if the custom client has never successfully called one of DH_RegisterPoint, DH_RegisterExistingPoint or DH_RegisterAllPoints. This is because the Cascade DataHub does not maintain internal state information for clients that are not registered to receive point exceptions.

See Also

DH_SetTransmitFormat

DH_SetTransmitFormat

DH_SetTransmitFormat — sets client-to-DataHub transmissions to binary or ASCII.

Syntax

```
#include <cogent.h>
int DH_SetTransmitFormat (
   int msgformat
   );
```

Arguments

msgformat

One of these:

PT_FMT_ASCII	Constructs a humanly readable ASCII string (the default).
PT_FMT_BINARY	Encodes data as a raw binary representation, efficient for parsing.
PT_FMT_HEX	Encodes data as ASCII encoded hex. This is not a valuable format since it is more difficult to parse and less space efficient than binary, and it is more difficult for humans to read than regular ASCII.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function sets a global variable that determines whether DH_WritePoint will attempt to use binary or ASCII format in transmissions from a client to the Cascade DataHub. It instructs the API to transmit all point change messages from the custom client to the DataHub using the specified message format.

See Also

DH_SetReceiveFormat

DH_UnregisterPoint

DH_UnregisterPoint — unregisters a point from the Cascade DataHub.

Syntax

```
#include <cogent.h>
ST_STATUS DH_UnregisterPoint (
   IP_Task* myself,
   PT_pCPOINT ppoint,
   IP_Msg* hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit.

ppoint

A pointer to a Cascade DataHub point structure, which must contain at least name. All other fields in the point structure should be 0 if this is the first call on this point, because they will be filled in by this call. Subsequent calls on the same point do not require zeroing any fields. If one point structure is used with several point names, the address field must be zeroed each time a different name is used in a call to this function.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function stops the Cascade DataHub from transmitting exceptions for the *ppoint*.

See Also

DH_RegisterPoint

DH_WritePoint, DH_WriteExistingPoint, DH_WriteExistingPoints

DH_WritePoint, DH_WriteExistingPoint, DH_WriteExistingPoints — write points to the Cascade DataHub.

Syntax

```
#include <cogent.h>
ST_STATUS DH_WritePoint (
   IP_hTASK myself,
   PT_pCPOINT ppoint,
   IP_hMSG hmsg,
   ER_hLIST elist
);
ST_STATUS DH_WriteExistingPoint (
   IP_hTASK myself,
   PT_pCPOINT ppoint,
   IP_hMSG hmsg,
   ER_hLIST elist
);
ST_STATUS DH_WriteExistingPoints (
   IP_hTASK myself,
   PT_pCPOINT ppoints,
   int npoints,
   IP_hMSG hmsg,
   ER_hLIST elist
);
```

Arguments

myself

The task handle associated with this task. This should always be the return value from IP_NserveInit

ppoint

A pointer to a Cascade DataHub point structure. All fields are valid, and will be treated by the write, but in the first call on a point the address field must be zero. It will be filled in automatically. This address will be valid so long as the point exists in the DataHub, and if, in subsequent DH_* functions, the address is provided for this point, the function will run slightly faster.

ppoints

A pointer to an array of DataHub points. Each of the points in the array must already exist in the DataHub.

npoints

The number of points in the array of points.

hmsg

A handle to a previously allocated message structure (using IP_MsgCreate) providing enough buffer space to handle the inter-process communication associated with the DataHub transaction.

elist

A return value containing error information. This is unimplemented in this version, and should be NULL.

Returns

ST_OK on success, ST_ERROR on failure, or some other value of ST_STATUS.

Description

This function writes a point into the DataHub. If the named point does not exist then DH_WritePoint will create it on the DataHub and DH_WriteExistingPoint will return the error ST_NO_POINT.

DH_WriteExistingPoints writes values to an array of existing points. It reduces processing time by sending as many points as possible in one message.

See Also

Communicating with the Cascade DataHub, DH_CreatePoint, DH_ReadPoint, DH_RegisterPoint

IV. Cogent Driver Functions

Table of Contents

DR_ApCloseIPC 89
DR_ApCommand 90
DR_ApConnectIPC 91
DR_ApDescribeBuffer 92
DR_ApDescribePnt 94
DR_ApInitIPC 96
DR_ApListBuffers 97
DR_ApListPoints 99
DR_ApPointBufAddress 101
DR_ApReadBlock 103
DR_ApReadControl 105
DR_ApReadPoint 107
DR_ApReadStatus 109
DR_ApUpdateBuffers 111
DR_ApWriteBlock 112
DR_ApWriteControl 114
DR_ApWritePoint 116

DR_ApCloseIPC

DR_ApCloseIPC — closes the IPC link.

Syntax

```
#include <cogent.h>
int DR_CloseIPC (
   );
```

Arguments

none

Returns

An integer value 0 if the IPC system terminated successfully, otherwise the following error may be reported:

```
DR_API_ERROR
```

Description

This function closes the IPC link to the driver.

Example

```
DR_ApCloseIPC ();
```

DR_ApCommand

DR_ApCommand — sends ASCII commands and returns replies.

Syntax

```
#include <cogent.h>
int DR_ApCommand (
   char* command,
   char* reply,
   int max_length,
   char** error
);
```

Arguments

command

A string containing the command to be sent.

reply

A pointer to the buffer to receive the reply from the driver.

max_length

The length of the reply buffer in bytes.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the command exchange was successful, otherwise the following errors may be reported:

```
DR_API_STATUS_ERRORS
DR_API_IPC_ERRORS
```

Description

This function sends an ASCII script command to the driver, returning the ASCII reply. You will need to interpret the response (see Configuration File in the CIF Driver manual). This function provides an all-purpose interface to the driver and is required infrequently.

Example

```
int  result;
char data[1024];
char *error_str;

if (!(result = DR_ApCommand ("(apropos *)", data, 1024,
                             &error_str)))
 printf ("   Driver commands available:\n%s\n", data);
else
 printf ("     Error@ApCommand (%d, %s)\n",
            result, error_str);
```

DR_ApConnectIPC

DR_ApConnectIPC — connects to Cogent products via IPC.

Syntax

```
#include <cogent.h>
int DR_ApConnectIPC (
   char* taskname,
   IP_hTASK my_task,
   IP_hMSG send_msg,
   IP_hMSG reply_msg,
   char* admin_name
);
```

Arguments

taskname

The name of the user's application; it can be any unique name in the system's namespace.

my_task

The task structure required to establish inter-process communication (IPC) with another Cogent product. If NULL is specified, then this function will create the required internal task structure. The process of creating the task structure will automatically publish the taskname to the Cascade NameServer (**nserve**) if it is running.

send_msg, reply_msg

The message structures required to send and receive IPC messages to another Cogent product. If NULL is specified, then this function will create the required internal structures with a maximum buffer size of 2048 bytes. Messages greater than 2048 bytes in length will be truncated.

admin_name

Taskname or PID of the driver (see drcif_ad in the respective Cogent Driver manual).

Returns

The integer value 0 (DR_API_OK) if the IPC system initialized successfully, otherwise the following error may be reported:

DR_API_ERROR

Description

This function permits users familiar with the Cogent library to connect to the driver using already defined task and/or message buffers. This is the underlying function in DR_ApInitIPC, which simply calls this function with NULL task and message buffer parameters.

DR_ApDescribeBuffer

DR_ApDescribeBuffer — gets segment attribute information.

Syntax

```
#include <cogent.h>
int DR_ApDescribeBuffer (
   int card_id,
   int buffer_id,
   int initial_seg,
   int max_attr,
   int* num_attr,
   DR_ApSegAttributes_t* attributes,
   char** error
);
```

Arguments

card-id

The card ID of the requested buffer.

buffer_id

The buffer ID of the requested buffer.

initial_seg

The ID of the first segment to be described. This parameter, in combination with *max_attr*, permits only a portion of the segments to be listed. This is useful if there are a large number of segments and limited memory space to contain them. Segments can be referenced as `0` to `n-1`.

max_attr

The maximum number of segments to be described, which must be less than or equal to the length of the array of `DR_ApSegAttributes_t` structures.

attributes

A pointer to an array of `DR_ApSegAttributes_t` structures, of length at least *max_attr*, that will receive the segment attribute information. Each element in the array (a segment attribute structure) describes a segment of the buffer with a particular data type and r/w capability. Multiple segments can exist in a buffer, so that more than one set of attributes may be returned for each buffer. Each segment description includes an offset in the buffer and length in bytes, the segment type and the readable and writeable attributes.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be `NULL` if no error string is required.

Returns

The integer value `0` if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_API_STATUS_ERRORS
DR_ERR_CARD_INVALID
DR_ERR_BLK_INVALID
DR_ERR_BLK_NOT_FOUND
DR_ERR_NO_USER_OBJECT
```

Description

This function gets information on the segment attributes of the specified buffer.

Example

```
int                     i, j, n_attr, num_blks;
DR_ApSegAttributes_t    seg_attr[4][32];

/* continued from example of DR_ApListBuffers */

for (i=0; i<num_blks; i++)
{
  if (!(result = DR_ApDescribeBuffer (0, i, 0, 32, &n_attr,
                      &(seg_attr[i][0]), &error_str)))
  {
      printf ("   Buffer %d:\n", i);
      for (j=0; j<n_attr; j++)
      {
          printf ("[%d] %s%s %s data from %d for %d bytes\n",
                  (seg_attr[i][j].buf_id),
                  (seg_attr[i][j].readable?"reads":""),
                  (seg_attr[i][j].writeable?"writes":""),
                  ((DR_API_BIT_TYPE == seg_attr[i][j].type)
                        ?"digital":
                  ((DR_API_INT16_TYPE ==seg_attr[i][j].type)
                        ?"analog":"unknown")),
                   seg_attr[i][j].offset,
                   seg_attr[i][j].size);
       }
  }
  else
    printf ("     Error@DescribeBuffer (%d,%s)\n",
            result, error_str);
}

printf ("   Digital read requires blocks: ");
for (i=0; i<num_blks; i++)
      for (j=0; j<n_attr; j++)
      {
            if (seg_attr[i][j].readable &&
                seg_attr[i][j].type==DR_API_BIT_TYPE)
            {
                  printf(" %d[%d] for %d, ",
                  seg_attr[i][j].buf_id,
                  seg_attr[i][j].offset,
                  seg_attr[i][j].size);
            }
       }
printf ("\n");

printf ("   Analog write requires blocks: ");
for (i=0; i<num_blks; i++)
      for (j=0; j<n_attr; j++)
      {
            if (seg_attr[i][j].writeable &&
                seg_attr[i][j].type==DR_API_INT16_TYPE)
            {
                  printf(" %d[%d] for %d, ",
                         seg_attr[i][j].buf_id,
                         seg_attr[i][j].offset,
                         seg_attr[i][j].size);
            }
       }
printf ("\n");
```

DR_ApDescribePnt

DR_ApDescribePnt — gives a description of a point.

Syntax

```
#include <cogent.h>
int DR_ApDescribePnt (
   char* pnt_name,
   int* pnt_type,
   int* enabled,
   int* readable,
   int* writeable,
   char* address,
   int addr_len,
   char** error
);
```

Arguments

pnt_name

A string containing the name of the required point.

pnt_type

If not NULL and the point is defined, then the type of the point is returned here. The following point types are defined (see dr_api.h):

* DR_API_DOUBLE_TYPE
* DR_API_INT16_TYPE
* DR_API_BIT_TYPE

enabled

If not NULL and the point is defined, then the status of the point is returned here, 1 if enabled, 0 if disabled.

readable

If not NULL and the point is defined, then 1 is returned if the point is configured as readable.

writeable

If not NULL and the point is defined, then 1 is returned if the point is configured as writeable.

address

If not NULL, the address of a string to be set to the point's address string, as specified during the configuration.

addr_len

The maximum length of the address string. Although addressing requirements vary with the type of driver and point, 16 characters is usually sufficient.

error

Address of a string pointer, in case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value `0` if the point was accessed successfully, otherwise one of the following error codes:

```
DR_ERR_PNT_NOT_FOUND
DR_ERR_PNT_NOT_ENABLED
DR_ERR_PNT_NOT_READABLE
DR_ERR_PNT_TYPE_NO_REP
DR_API_STATUS_ERRORS
DR_API_IPC_ERRORS
```

Description

This function gets a description of the named point.

Example

```
int         enabled, readable, writeable, type;
char        *name = "counter";
char        address[16];
char        *error_str;

if result = DR_ApDescribePoint (name, &type, &enabled,
                                &readable, &writeable,
                                address, sizeof(address), &error_str))
{
        printf ("     Error@DescribePoint (%d, %s)\n",
                result, error_str);
}
else
{
        printf ("     Point %s: type %d, %s%s, addr: %s\n",
                test_pnt, type, (readable?"R":""),
                (writeable?"W":""), address);
}
```

DR_ApInitIPC

DR_ApInitIPC — opens the connection to the driver.

Syntax

```
#include <cogent.h>
int DR_ApInitIPC (
   char* taskname,
    char* admin_name
);
```

Arguments

taskname

The name of the user's application. It can be any unique name in the system's namespace.

admin_name

The taskname of the driver (see drcif_ad in the respective Cogent Driver manual). The driver PID can also be used.

Returns

The integer value 0 (DR_API_OK) if the IPC system initialized successfully, otherwise the following error may be reported:

DR_API_ERROR

Description

This function opens the connection to the driver. If the Cascade NameServer (**nserve**) is available, then it is used to locate the driver by name, otherwise the QNX **nameloc** service is used. Send and reply message buffers are created with a default size of 2048 bytes. Subsequent IPC messages greater than 2048 bytes in length will be truncated.

Example:

```
char  *taskname = "dr_test";
char  *admin_name = "/dr/cif";

if (argc > 1)
{
        taskname = argv[1];
        if (argc > 2)
           admin_name = argv[2];
}

if (!DR_ApInitIPC (taskname, admin_name))
{
        printf ("Successfully connected %s to %s\n\n",
                 taskname, admin_name);
}
else
        printf ("Failed to connect %s to %s\n", argv[0], argv[1]);
```

DR_ApListBuffers

DR_ApListBuffers — lists blocks defined to the driver.

Syntax

```
#include <cogent.h>
int DR_ApListBuffers (
   int card_id,
   int max_bufs,
   int* num_bufs,
   unsigned short* size,
   char** error
);
```

Arguments

card_id

The card ID of the requested block of data.

max_bufs

The maximum number of buffers to be listed.

num_bufs

If not NULL, the address of the variable to return the actual number of buffers associated with the specified card.

size

If not NULL, a pointer to an array at least *max_bufs* long, that will be filled with the size (in bytes) of each of the *num_bufs* blocks.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_API_STATUS_ERRORS
DR_ERR_CARD_INVALID
DR_ERR_BLK_INVALID
DR_ERR_BLK_NOT_FOUND
DR_ERR_NO_USER_OBJECT
```

Description

This function provides a list of the blocks currently defined to the driver. This information may then be used to get further block attribute information and read or write the blocks.

Example

```
int             i, num_blks, result;
unsigned short  buf_size[4];
char            *error_str;

num_blks = 0;
if (!(result = DR_ApListBuffers (0, 4, &num_blks, buf_size,
                                 &error_str)))
{
  printf ("   Device 0 has %d buffers with sizes ", num_blks);
        for (i=0; i<num_blks; i++)
              printf ("%d%s", buf_size[i],
                      ((i+1==num_blks)?".":", "));
        printf ("\n");
}
else
        printf ("     Error@ListBuffers (%d,%s)\n",
                result, error_str);
}
```

DR_ApListPoints

DR_ApListPoints — lists points defined to the driver.

Syntax

```
#include <cogent.h>
int DR_ApListPoints (
   char* filter,
   int* num_names,
   char** pnt_names,
   int max_names,
   char* data,
   int max_data_length,
   char** error
);
```

Arguments

filter

A string containing a filter specification for the points, including name pattern, and point status and type. If NULL is provided, then all points will be requested (the equivalent of using a filter pattern of "*"). The filter syntax is as follows:

```
name_pattern [w][r][e][d][t type]
```

where:

- *name_pattern* is required and supports wildcard characters * and ?.
- The optional status filters w, r, e, and d correspond to the points writeable, readable, enabled, and disabled status respectively.
- The optional type filter t must be followed by a valid type name: digital, integer, group, string, heartbeat or real.

num_names

If not NULL, then the number of names found is returned here.

pnt_names

If not NULL, a pointer to an array of char pointers (at least *max_names* long), to be filled with pointers to the name strings returned by the driver in the data buffer. The pointers will point to parsed data contained in the buffer specified by the *data* parameter (if provided). Otherwise the data will remain in the internal message buffers (referenced by the char pointers) and can only be guaranteed to remain valid until the next API function call.

max_names

The length of the *pnt_names* array (maximum number of names that can be returned).

data

If not NULL, then the string of point names returned by the driver is copied to this buffer, prior to parsing and assigning name pointers. Since the point names are returned as an array of pointers to a parsed buffer, this parameter provides a mechanism to ensure the contents of *pnt_names* will continue to be valid beyond the next API function call. In some cases, you may wish to simply check or explicitly copy out just a few names, and not require that the *names* data persist. In these cases, copying the data is not required, and a NULL can be supplied for this parameter.

max_data_length

The length of the data buffer in bytes.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be `NULL` if no error string is required.

Returns

The integer value `0` if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_API_STATUS_ERRORS
```

Description

This function provides a list of the points currently defined to the driver. The result can be used to determine the individual point characteristics.

Example

```
int         j, nargs, type, result;
char        data[1024];
char        *nargv[128];
char        *error_str;

result = DR_ApListPoints (NULL, &nargs, nargv, 128, data, 1024,
                          &error_str);
if (!result)
{
        for (j = 1; j<nargs; j++)
        {
                DR_ApReadPoint(nargv[j], &type, &value, &error_str);
                /* process point */
        }
}
```

DR_ApPointBufAddress

DR_ApPointBufAddress — deletes a point's image buffer address.

Syntax

```
#include <cogent.h>
int DR_ApPointBufAddress (
   char* pnt_name,
   int* card_id,
   int* buf_id,
   unsigned short* offset,
   unsigned short* bit,
   char** error
);
```

Arguments

pnt_name

A string containing the name of the required point.

card_id

If not NULL and the point is defined, returns the number of the card providing access to the point data.

buf_id

If not NULL and the point is defined, returns the ID of the buffer containing the point data.

offset

If not NULL and the point is defined, returns the byte offset into the buffer for the point data.

bit

If not NULL, the point is defined, and the point is of digital (bit) type, returns the bit value of the word corresponding to the point data.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the function is successful, otherwise one of the following error codes:

```
DR_ERR_PNT_NOT_FOUND
DR_API_IPC_ERRORS
DR_API_STATUS_ERRORS
```

Description

This function determines the image buffer address associated with the named point.

Example

```
int             card, buffer;
unsigned short  offset, bit;
char            *name = "counter";
char            *error_str;

if (result = DR_ApPointBufAddress (name, &card, &buffer,
                                   &offset, &bit, &error_str))
        printf ("     Error@PointBufAddress (%d, %s)\n",
                result, error_str);
else
{
        printf ("     Maps to: card %d, buffer %d, offset: %d",
                card, buffer, offset);
        printf (", bit: %d\n", bit);
}
```

DR_ApReadBlock

DR_ApReadBlock — reads a block.

Syntax

```
#include <cogent.h>
int DR_ApReadBlock (
   int card_id,
   int buf_id,
   unsigned short offset,
   unsigned short size,
   void* data,
   char** error
);
```

Arguments

card_id

The card ID of the requested block of data.

buf_id

The buffer ID of the requested block of data.

offset

The starting (byte) address of the requested block of data into the specified card buffer.

size

The number of bytes to be read.

data

The address of memory area, at least *size* byes long, to receive data.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_ERR_CARD_INVALID
DR_ERR_BLK_INVALID
DR_ERR_BLK_OFS_INVALID
DR_ERR_BLK_SIZE_INVALID
DR_ERR_CMD_INVALID
```

Description

This function reads the specified block.

Example

```
char    data[1024];
int     result, i;
char    *error_str;

printf ("   Reading output block\n");
if (result = DR_ApReadBlock (0, 0, 0, 64, data, &error_str))
      printf ("     Error@ReadBlock (%d,%s)\n",
              result, error_str);
else
{
       dump_data (6, data, 8, 16);
}
```

DR_ApReadControl

DR_ApReadControl — gets card control parameters.

Syntax

```
#include <cogent.h>
int DR_ApReadControl (
   int card_id,
   int buffer,
   unsigned short offset,
   unsigned short size,
   void* status,
   char** error
);
```

Arguments

card_id

The ID of the card to be read for control parameters.

buffer

The ID of the target control parameter buffer (if applicable).

offset

The starting (byte) address for the specified data within the card buffer (if applicable).

size

The maximum number of bytes to be written.

status

A pointer to a card-dependent structure (at least *size* bytes long) that contains the control parameter information for the card.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_ERR_CARD_INVALID
DR_ERR_BLK_INVALID
DR_ERR_BLK_OFS_INVALID
DR_ERR_BLK_SIZE_INVALID
DR_ERR_CMD_INVALID
```

Description

This function obtains the current card control parameters. The control parameter data is dependent on the card.

Example

```
cif_ApParms_t control;

if (result = DR_ApReadControl (0, 2, 0, sizeof (cif_ApParms_t),
                               &control, &error_str))
        printf ("Error@ReadControl (%d,%s)\n", result, error_str);
else
{
        printf ("Mode = %X, cycleTime = %d, Storage format = %d\n",
                control.Mode, control.CycleTime, control.Format);
}
```

See Also

Hilscher Fieldbus CIF Card in the Cogent C API manual.

DR_ApReadPoint

DR_ApReadPoint — reads the value of a point.

Syntax

```
#include <cogent.h>
int DR_ApReadPoint (
   char* pnt_name,
   int* pnt_type,
   DR_ApValue_t* pnt_value,
   char** error
);
```

Arguments

pnt_name

A string containing the name of the required point.

pnt_type

A pointer to an integer to return the point type code. This is needed to access the *pnt_value*. The list of defined driver data point types is described in the Overview section of the Device Driver for Hilscher Fieldbus Cards manual. This parameter can be NULL if the type of point is already known.

pnt_value

A pointer to the memory area to receive the point value. *pnt_value* is a union of the different types, and *pnt_type* is used to access the data.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the point was accessed successfully, otherwise one of the following error codes is returned:

```
DR_ERR_PNT_NOT_FOUND
DR_ERR_PNT_NOT_ENABLED
DR_ERR_PNT_NOT_READABLE
DR_ERR_PNT_TYPE_NO_REP
DR_API_STATUS_ERRORS
DR_API_IPC_ERRORS
```

Description

This function reads the current value of the named point.

Example

```
int             type, result;
DR_ApValue_t    value;
char            *error_str, *name = "Pushbutton";

result = DR_ApReadPoint (name, &type, &value, &error_str);
if (result == 0)
{
```

```
   printf ("     Pnt: %s ", name);
 switch(type)
 {
      case DR_API_DOUBLE_TYPE:
          printf ("(real) = %f\n", value.d);
          break;
      case DR_API_INT32_TYPE:
          printf ("(int) = %d\n", value.i);
          break;
      case DR_API_INT16_TYPE:
          printf ("(short) = %d\n", value.s);
          break;
      case DR_API_BIT_TYPE:
          printf ("(bit) = %1X\n", value.s);
          break;
      default:
          printf ("(%d) = %X\n", type, value.i);
 }
}
else
      printf ("     Error@ListPoints:%s (%d, %s): \n",
              name, result, error_str);
```

DR_ApReadStatus

DR_ApReadStatus — gets card status information.

Syntax

```
#include <cogent.h>
int DR_ApReadStatus (
   int card_id,
   int buffer,
   unsigned short offset,
   unsigned short size,
   void* status,
   char** error
);
```

Arguments

card_id

The ID of the card to be read for status.

buffer

The ID of the requested status buffer (if applicable).

offset

The starting (byte) address of the required data within the specified card buffer (if applicable).

size

The maximum number of bytes to be read.

status

A pointer to a card-dependent structure (at least *size* bytes long) that will contain the status information for the card.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_ERR_CARD_INVALID
DR_ERR_BLK_INVALID
DR_ERR_BLK_OFS_INVALID
DR_ERR_BLK_SIZE_INVALID
DR_ERR_CMD_INVALID
```

Description

This function obtains the card status information. The status data is dependent on the card.

Example

```
cif_ApState_t status;

if (result = DR_ApReadStatus (0, 2, 0, sizeof (cif_ApState_t),
                              &status, &error_str))
        printf ("Error@ReadStatus (%d,%s)\n", result, error_str);
else
{
        printf ("     GlobalState = %s, %s, %s\n",
                (status.GlobalBits & STATE_GLOBAL_CTRL ?
                     "Parm error" : "Parms OK"),
                (status.GlobalBits & STATE_GLOBAL_ACLR ?
                     "AutoClear/Slave error" : "Mode/Slaves OK"),
                (status.GlobalBits & STATE_GLOBAL_NDATA ?
                     "Slave Fatal error" : "Slaves OK"));
        printf ("     DPMState = %2.2X %s\n", status.DPMState,
                (status.DPMState ?
                (status.DPMState & STATE_OPERATE ? "OPERATING":
                (status.DPMState & STATE_STOP ?"STOPPED":
                (status.DPMState & STATE_CLEAR ? "CLEAR" :
                      "Unknown state"))):"OFFLINE"));
        printf ("     Error status: (%d, %d)\n",
                 status.ErrorRemoteAddr, status.ErrorEvent);
        printf ("     Slaves state: ");
        for (i=0; i<16; i++)
                 printf ("%2.2X", status.SlaveState[i]);
        printf ("\n");
        printf ("     Slaves diagnostic bits: ");
        for (i=0; i<16; i++)
                 printf ("%2.2X", status.SlaveDiags[i]);
        printf ("\n");
}
```

See Also

Hilscher Fieldbus CIF Card in the Cogent C API manual.

DR_ApUpdateBuffers

DR_ApUpdateBuffers — forces an I/O cycle of the image buffers.

Syntax

```
#include <cogent.h>
int DR_ApUpdateBuffers (
   );
```

Arguments

none

Returns

The integer value 0 if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_API_STATUS_ERRORS
```

Description

This function forces an I/O cycle of the image buffers to and from the field cards. The output buffers are written to the output field cards, and the input buffers are updated with data from the input field cards.

Example

```
DR_ApUpdateBuffers ();
```

DR_ApWriteBlock

DR_ApWriteBlock — writes data to a block.

Syntax

```
#include <cogent.h>
int DR_ApWriteBlock (
   int card_id,
   int buf_id,
   unsigned short offset,
   unsigned short size,
   void* data,
   char** error
);
```

Arguments

card_id

The card ID of the requested block of data.

buf_id

The buffer ID of the requested block of data.

offset

The starting (byte) address of the requested block of data into the specified card buffer.

size

The number of bytes to be written.

data

The address of memory area, at least *size* byes long, containing the data.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_ERR_CARD_INVALID
DR_ERR_BLK_INVALID
DR_ERR_BLK_OFS_INVALID
DR_ERR_BLK_SIZE_INVALID
DR_ERR_CMD_INVALID
```

Description

This function writes data to the specified block.

Example

```
/* continued from example in DR_ApReadBlock */
printf ("   Write Block Bit Cycle:\n", 1);
data[32] = 0;
for (i=0; i<=8 && result==0; i++)
{
        if (data[32] == 0)
              data[32] = 1;
        else
              data[32] *= 2;
        printf ("\r      %4X ", data[32]);
        fflush (stdout);
        if (result = DR_ApWriteBlock (0, 0, 0, 64, data,
                                      &error_str))
        printf ("     Error@WriteBlock (%d,%s)\n",
                result, error_str);
        delay (250);
}
printf ("\n");
```

DR_ApWriteControl

DR_ApWriteControl — sends card control information to the driver.

Syntax

```
#include <cogent.h>
int DR_ApWriteControl (
   int card_id,
   int buffer,
   unsigned short offset,
   unsigned short size,
   void* control,
   char** error
);
```

Arguments

card_id

The ID of the card to be written with status.

buffer

The ID of the target status buffer (if applicable).

offset

The starting (byte) address of the status data within the specified card buffer (if applicable).

size

The maximum number of bytes to be written.

control

A pointer to a card-dependent structure (at least *size* bytes long) that contains the status information for the card.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the function was successful, otherwise one of the following error codes:

```
DR_API_IPC_ERRORS
DR_ERR_CARD_INVALID
DR_ERR_BLK_INVALID
DR_ERR_BLK_OFS_INVALID
DR_ERR_BLK_SIZE_INVALID
DR_ERR_CMD_INVALID
```

Description

This function sends card control information to the driver. The control data is dependent on the card.

Example

```
cif_ApParms_t control;

control.Mode = mode;
control.CycleTime = ms;
control.Format = fmt;
DR_ApWriteControl (0, 2, 0, sizeof (cif_ApParms_t),
                   &control_parms, NULL );
```

See Also

Hilscher Fieldbus CIF Card in the Cogent C API manual.

DR_ApWritePoint

DR_ApWritePoint — writes a new value to a point.

Syntax

```
#include <cogent.h>
int DR_WritePoint (
   char* pnt_name,
   int pnt_type,
   DR_ApValue_t* pnt_value,
   char** error
);
```

Arguments

pnt_name

A string containing the name of the required point.

pnt_type

The point type code. The list of defined driver data point types is described in the Overview section of the Device Driver for Hilscher Fieldbus Cards manual.

pnt_value

A pointer to the memory area containing the point value, of the same type as specified in *pnt_type*.

error

The address of a string pointer. In case of error (non-zero return), the string pointer is set to the corresponding error description string. The error string is contained in a static buffer and remains valid only until the next API call. The parameter may be NULL if no error string is required.

Returns

The integer value 0 if the point was updated successfully, otherwise the following error code is returned:

DR_ERR_PNT_NOT_WRITEABLE

Description

This function writes a new value to the named point. The field I/O is updated immediately, unless the point is a member of a writeable group, in which case it is updated when the group is processed.

Example

```
int             result;
DR_ApValue_t    value;
char            *error_str, *name = "Red_Light1";

value.s = 1;
result = DR_ApWritePoint (name, DR_API_BIT_TYPE, &value,
                          &error_str);
```

See Also

DR_ApReadPoint

V. Cascade Historian Functions

Table of Contents

HI_Add 118
HI_BufferIDDestroy 120
HI_BufferIDLength 121
HI_BufferIDRead 122
HI_Bufsize 124
HI_ClipBuffer 126
HI_Count 127
HI_Deadband 128
HI_Delete 131
HI_Describe 132
HI_Disable 134
HI_Earliest 135
HI_Enable 136
HI_ExchangeBuffer 137
HI_FileBase 138
HI_Flush 140
HI_GapCountBuffer 141
HI_GapFillBuffer 142
HI_History 143
HI_Interpolate 145
HI_InterpolateData 148
HI_Latest 150
HI_Length 151
HI_List 152
HI_ScaleBuffer 154
HI_StatBuffer 155
HI_Register 156
HI_Unregister 157
HI_Version 158

HI_Add

HI_Add — adds a single value at a specified time.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Add (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname,
   double value,
   time_t seconds,
   time_t nanoseconds
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of a history.

value

The value to be added to the history.

seconds

The time in seconds, normally seconds since Jan 1, 1970, GMT.

nanoseconds

The nanoseconds within the second.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you add points and their values manually to the Cascade Historian. It injects a single value at the specified time, in seconds and nanoseconds. This is useful for sending data directly to the historian (without using Cascade Datahub) as well as for testing and debugging, where one or more points' values can be changed in a controlled way.

In order to guarantee the time-series data is ordered, the time of the previous data point is checked and if a time reversal is detected (i.e., the current timestamp preceeds that of the last one) then the timestamp of the data is modified to a small increment (1 micro-second) relative to the previous timestamp. This

situation should only occur when there is more than one source of the same data, a situation that is never recommended.

The value for the point is stored internally as a `HI_stVALUE` structure, where `HI_stVALUE.xaxis` is the time, represented as a double-precision floating point number, and `HI_stVALUE.value` is the value.

Since the Cascade Historian handles points sequentially on a time-stamped basis, if you call this function several times on the same point each call should have increasing *`seconds`* and/or *`nanoseconds`* parameters.

This function corresponds to the Cascade Historian **add** command.

HI_BufferIDDestroy

HI_BufferIDDestroy — destroys an interpolation buffer.

Syntax

```
#include <cogent.h>
ST_STATUS HI_BufferIDDestroy (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   int bufferid
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

bufferid

The buffer identifier, as created by a call to HI_Interpolate.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function destroys the temporary internal buffer associated with an interpolation request. The memory allocated by an interpolation request can be very large, and will persist until it is explicitly destroyed.

This function corresponds to the Cascade Historian **bufferIdDestroy** command and the hist_buffer_id_destroy dynamic library function.

HI_BufferIDLength

HI_BufferIDLength — gives the length of an interpolation buffer.

Syntax

```
#include <cogent.h>
ST_STATUS HI_BufferIDLength (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   int bufferid,
   int* length
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

bufferid

The buffer identifier, as created by a call to HI_Interpolate.

length

Return value containing the number of values in an interpolation buffer.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function determines the exact number of values in the interpolation buffer identified by *bufferid*, and returns that number in *length*.

This function corresponds to the Cascade Historian **bufferIdLength** command and two dynamic library functions: hist_buffer_id_length and hist_length_buffer.

HI_BufferIDRead

HI_BufferIDRead — reads an interpolation buffer.

Syntax

```
#include <cogent.h>
ST_STATUS HI_BufferIDRead (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   int bufferid,
   int start,
   int count,
   HI_stVALUE* value
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

bufferid

The buffer identifier, as returned by a call to HI_Interpolate.

start

An offset into the buffer for the returned data.

count

The number of values to be read.

values

An array of HI_stVALUEs with at least *count* elements.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function is used for reading the contents of a buffer created by HI_Interpolate. You are responsible for defining an array (or allocating the memory) to contain the *values* read. The length of this array can be determined with HI_BufferIDLength.

Some queries can result in large buffers whose size makes it impractical to read with a single IPC message. The default IPC message buffer used by the Cogent C API sets a limit of about 200 values. For higher numbers of values, the *start* and *count* parameters can be used to read the buffer in segments.

This function corresponds to the Cascade Historian **bufferIdData** command and the `hist_buffer_id_read` dynamic library function.

HI_Bufsize

HI_Bufsize — sets the history buffer size.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Bufsize (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname,
   int numvalues,
   int size
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of a history.

numvalues

The new maximum number of value/time (or y/x) pairs that can be buffered for this history. If 0 is specified, then the function is treated as a query, the buffer size is not modified, and the current value is returned in *size*.

size

A pointer to an integer variable that is set to the current maximum number of value/time (or y/x) pairs that can be buffered for this history.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function sets the number of data values that will be buffered for the specified history. (There are actually up to twice this number stored at any time, as the Cascade Historian double-buffers its data storage.) When a buffer is full, the data is flushed to disk, as long as an associated file has been assigned (with **filebase** or HI_FileBase). A larger buffer will retain more data in memory and increase the speed of queries on that data if it is within the range of the data in the buffer.

The history data is flushed to disk whenever this function is called. Until this function is used, the history buffer size is set to the initial default value (100).

This function corresponds to the Cascade Historian **bufsize** command and the `hist_bufsize` dynamic library function.

HI_ClipBuffer

HI_ClipBuffer — clips Y values to fit within a range.

Syntax

```
#include <cogent.h>
ST_STATUS HI_ClipBuffer (
   int nvalues,
   HI_stVALUE* values,
   double min,
   double max
);
```

Arguments

nvalues

The number of values in the *values* array.

values

An array of HI_stVALUE structures.

min

The minimum of the clipping range (Y values less than this value will be set to this value).

max

The maximum of the clipping range (Y values greater than this value will be set to this value)

Returns

t (true) if the function completed successfully, otherwise nil.

Description

This function clips the Y-value data in a buffer to lie within the range specified. The X-value data is ignored. The buffer is modified in place (i.e., a new buffer is not created, and the old values cannot be recovered). This function is useful during graphing to ensure the data trace does not go outside the desired graph boundaries.

This function corresponds to the Cascade Historian hist_clip_buffer dynamic library function.

Example

```
HI_ClipBuffer (20, test, target + range, target - range);
```

HI_Count

HI_Count — counts the number of histories that match a pattern.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Count (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histpattern,
   int* count
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histpattern

A globbing pattern specifying a group of histories.

count

Return value containing the number of histories.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function queries the Cascade Historian for the total number of histories currently being kept that match the given pattern.

This function corresponds to the Cascade Historian **count** command and the hist_count dynamic library function.

HI_Deadband

HI_Deadband — sets the history value deadband.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Deadband (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname,
   char* flag|type,
   double setting
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of a history.

flag

A deadband flag, any of: enable or prior, as described below.

type

A deadband type, any of: absolute, percent, timelimit, or countlimit, as described below.

setting

A setting (possibly optional) corresponding to the specified *type* or *flag*. Settings for *flag*s are 0 (0.0) for true and any non-zero value for false.

Returns

ST_OK on success, and the *retbuf* may be parsed to extract the complete response from the Cascade Historian (see the **deadband** command). Otherwise ST_ERROR, and the retbufa will contain a NULL-terminated character string with an error message.

Description

This function sets a deadband on a history, such that new values falling within that deadband are not recorded.

A deadband is used to reduce the amount of data stored by only storing data if there is a significant change in value. This approach is superior to simply reducing the sampling frequency, which will lose information when data changes quickly, and will waste storage by saving the same values when data

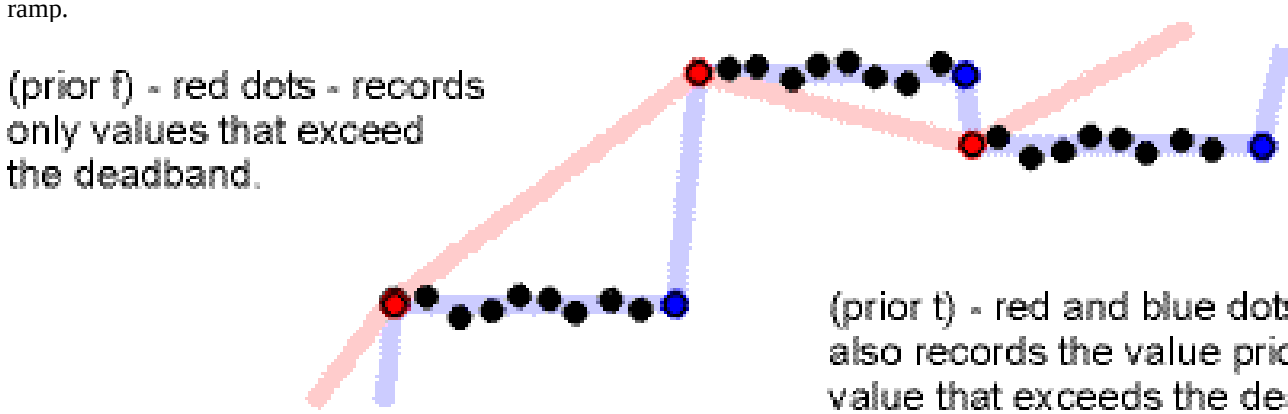
doesn't change. The deadband approach defines a resolution below which changes in data are deemed to be 'noise' and therefore ignored.

Flags

- **enable** turns the deadband on and off without affecting its configuration.

  In order for this flag to be set to true, at least one *type* must be set to a non-zero value. Otherwise, it will return false.

- **prior** records the value prior to any value that exceeds the deadband. If set to true (the default), when a value is encountered that exceeds the deadband, the value immediately prior to that is also recorded in the data set. This is done so that plotted data will be approximately correct. For example, if a value remains in a stable range for a long time, and then suddenly spikes to a large number, it is appropriate to keep the last known value within the deadband range before adding the spike value to the data set. When this data is plotted, the spike will actually show as a spike, rather than a gradual ramp.



Types

- **absolute** sets the deadband range to a single value. Any value differing from the baseline by less than `absolute` will not be recorded. If a value's difference from the baseline is greater than or equal to `absolute`, that value is recorded and it becomes the new baseline for the `absolute` deadband.
- **percent** sets the deadband range to be a percent of the last logged data value. Every new value is compared to that value, and if their percentage difference is less than `percent`, the new value will not be recorded. If the difference is greater than or equal to `percent`, the value is recorded and that value becomes the new baseline for the `percent` deadband.

  If `absolute` and `percent` are used together there is an AND relationship between them. The Cascade Historian will ignore any value falling within *either* deadband. Only those values falling outside all deadbands (or equal to the outermost) will be recorded.

- **timelimit** sets the maximum time period (in seconds, a real number) to deadband. When the `timelimit` is reached, the next new value received will be recorded, even if its value doesn't exceed the deadband. Note that the system does not automatically generate and insert a value when the `timelimit` is reached; it waits for the next new value. Whenever a new value is recorded the `timelimit` is restarted. If the `timelimit` parameter is not used, there is no time limit on how many values are ignored within a deadband.
- **countlimit** sets a maximum number of received values to deadband. When the `countlimit` is reached, the next new value will be recorded, even if it doesn't exceed the deadband. Whenever a new value is recorded the `countlimit` is restarted. If the `countlimit` parameter is not used, there is no count limit on how many values are ignored within a deadband.

This function corresponds to the Cascade Historian **deadband** command and the `hist_deadband` dynamic library function.

HI_Delete

HI_Delete — removes a history from memory only.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Delete (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histpattern
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histpattern

A globbing pattern specifying a group of histories.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function removes all memory associated with a given history, but does not remove any files on disk. It flushes the history to disk.

This function corresponds to the Cascade Historian **delete** command and the hist_delete dynamic library function.

HI_Describe

HI_Describe — describes a history's current configuration.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Describe (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

A *required* buffer containing an error message.

buflen

The length in bytes of *retbuf*, with a minimum as described below.

histname

The name of a history.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message.

Description

This function returns a description of the current configuration of a history in *retbuf*, which is not optional.

The information returned will be truncated at 1000 bytes.

The format of the return is:

```
(describe "histname" "directory" "basename" "extension" ndigits
"full_filename" flags deadband maxvalues)
```

Any undefined string values are returned as "", not the string "nil". The following values apply specifically:

- The *histname* as previously set with the **history** command, or the HI_History function.
- The *directory*, *basename*, *extension* and *ndigits* as previously set with the **filebase** command, or the HI_FileBase function.
- The *full_filename* indicates the pathname of the current file to which the data will be written, as set up by the four **filebase** parameters mentioned above.
- The *flags* are given in hexadecimal format (ie. 0x0064). The following constants define the individual flag bits and may be used to test the flags value:

Constant	Comment
HIST_DISABLED	Indicates history logging state.
HIST_REGISTERED	Indicates history is registered with the Cascade DataHub for value updates.
HIST_DEADBAND	Overall deadband state (on/off)
HIST_DEADBAND_FORCE	Internal state bit.
HIST_DEADBAND_PRIOR	Indicates PRIOR mode is set.
HIST_DEADBAND_PERCENT	Indicates PERCENT deadband is set.
HIST_DEADBAND_ABSOLUTE	Indicates ABSOLUTE deadband is set.
HIST_DEADBAND_TIMELIMIT	Indicates TIMELIMIT deadband is set.
HIST_DEADBAND_COUNTLIMIT	Indicates COUNTLIMIT deadband is set.
HIST_DEADBAND_TYPES	Bit-OR of the deadband type flags (PERCENT, ABSOLUTE, TIMELIMIT, and COUNTLIMIT, useful for testing if any deadband type is set.

- The status of the *deadband*, t (indicating on) or nil (indicating off). The **deadband** command (or HI_Deadband API function or hist_deadband DLL function) should be used to determine the actual deadband configuration.
- The *maxvalues* as previously set with the **bufsize** command, or the HI_Bufsize function.
- The *point* associated with the history, as previously set with the **history** command, or the HI_History function.

The *buflen* must be at least the sum of:

```
  strlen(histname)
+ (strlen(directory) * 2)
+ (strlen(basename) * 2)
+ (strlen(extension) * 2)
+ (strlen(full_filename) * 2)
+ 64
```

This function corresponds to the Cascade Historian **describe** command and the hist_describe dynamic library function..

HI_Disable

HI_Disable — stops data recording.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Disable (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histpattern
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histpattern

A globbing pattern specifying a group of histories.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function flushes one or more histories to disk and causes each of them to ignore all subsequent values until either **enable**, hist_enable, or HI_Enable is called on it.

This function corresponds to the Cascade Historian **disable** command and the hist_disable dynamic library function.

HI_Earliest

HI_Earliest — gives the earliest value in a history.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Earliest (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname,
   HI_stVALUE* data
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of a history.

data

The value and time of the first data item in the history.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function returns the value for the first data item in the history. This item may be either in a file or in memory.

This function corresponds to the Cascade Historian **earliest** command and the hist_earliest dynamic library function.

HI_Enable

HI_Enable — activates data recording.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Enable (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histpattern
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histpattern

A globbing pattern specifying a group of histories.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function reverses the effect of a previous **disable**, hist_disable, or HI_Disable call, so that new data values will be accepted and logged.

This function corresponds to the Cascade Historian **enable** command and the hist_enable dynamic library function.

HI_ExchangeBuffer

HI_ExchangeBuffer — swaps X and Y values.

Syntax

```
#include <cogent.h>
ST_STATUS HI_ExchangeBuffer (
   int nvalues,
   HI_stVALUE* values
);
```

Arguments

nvalues

The number of values in the *values* array.

values

An array of HI_stVALUE structures.

Returns

t (true) if the transformation was performed with no errors, otherwise nil.

Description

This function swaps the X and Y values for each data pair in the buffer. The buffer is modified in place (i.e., a new buffer is not created, and the old values cannot be recovered).

This function is used to obtain certain data queries that are not directly supported by the Cascade Historian's interpolation services. For example, if we want to interpolate x (typically time or position) at all actual (non-interpolated) values of y, the Cascade Historian does not directly support this, since the interpolator services only interpolate the Y value. The desired result can be obtained by making the inverse query and exchanging the buffer results.

This function corresponds to the Cascade Historian hist_exchange_buffer dynamic libary function.

HI_FileBase

HI_FileBase — associates the history with a file for data storage.

Syntax

```
#include <cogent.h>
ST_STATUS HI_FileBase (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname,
   char* dir,
   char* basename,
   char* ext,
   int numdigits
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of a history.

dir

The name of an existing directory, either absolute or relative to the directory in which **histdb** was started.

basename

The base file name, without an extension or directory.

ext

The file extension.

numdigits

An integer that specifies how the filename is augmented for multiple files, described in detail below.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This command associates a file with a history. Without an associated file, data received by the Cascade Historian will be lost when the buffer is full, or when the Cascade Historian is shut down. A history without a file (and with a sufficiently large buffer) could be used for temporary storage and analysis of data.

A history is potentially logged to multiple files. These can be numbered sequentially, or based on date, as specified by the *numdigits* parameter. The filename specified by the user is automatically augmented as each new file is opened, and the Cascade Historian understands that these files all belong together as a single history.

When **filebase** or HI_FileBase is first called for a history, it sets the pattern for all file names to be created for the history. Once this has been done, the Cascade Historian will be able to find the files relating to that history at any time, and will begin numbering from the next file number as necessary.

The possible values for the *numdigits* parameter, with their implications, are as follows:

- < 0 : Use the date to generate the unique suffix.
- = 0 : No sequential digits are added to the name. This is the default.
- > 0 : Specifies the number of decimal digits to be added to the filebase.

For example, the call:

```
HI_FileBase (hist, NULL, 0, "hist1", "/tmp/histories", "test.", ".dat", 2);
```

would create files of the form:

```
/tmp/histories/test.01.dat /tmp/histories/test.02.dat
/tmp/histories/test.03.dat ...
```

This function corresponds to the Cascade Historian **filebase** command and the hist_filebase dynamic library function.

HI_Flush

HI_Flush — writes buffered data to disk.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Flush (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histpattern
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histpattern

A globbing pattern specifying a group of histories.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function removes the data for the specified history or histories from the buffer and writes them to the associated files. If NULL or * are passed for *histpattern*, all buffers are flushed to file.

This function corresponds to the Cascade Historian **flush** command and the hist_flush dynamic library function.

HI_GapCountBuffer

HI_GapCountBuffer — identifies and counts large gaps in data.

Syntax

```
#include <cogent.h>
ST_STATUS HI_GapCountBuffer (
   int nvalues,
   HI_stVALUE* values,
   double gap
);
```

Arguments

nvalues

The number of values in the *values* array.

values

An array of HI_stVALUE structures.

gap

The threshold difference between adjacent X values that determines the existence of a gap.

Returns

The number of gaps, or (error).

Description

This function identifies and counts gaps for the HI_GapFillBuffer function. These two functions, when used together, correspond to the Cascade Historian hist_gap_buffer dynamic library function.

Example

See HI_GapFillBuffer

HI_GapFillBuffer

HI_GapFillBuffer — fills in data gaps found by HI_GapCountBuffer.

Syntax

```
#include <cogent.h>
ST_STATUS HI_GapFillBuffer (
   int nvalues,
   HI_stVALUE* values,
   int n2,
   HI_stVALUE* new_values,
   double gap
);
```

Arguments

nvalues

The number of values in the *values* array.

values

An array of HI_stVALUE structures.

n2

The size of the *new_values* array.

new_values

A new array of HI_stVALUEs to contain the data from *values*, with duplicate values inserted where the difference in time exceeds *gap*.

gap

The threshold difference between adjacent X values that determines the existence of a gap.

Returns

t (true) if the function completed successfully, otherwise nil.

Description

This function inserts duplicate Y-value points into the buffer when the X-axis gap between a pair of values exceeds the specified gap threshold. This will cause the graph of the data to show a constant value spanning the gap, rather than a misleading ramp between the points.

This function requires the return value from HI_GapCountBuffer to calculate and allocate the size of the new array that will contain the additional data. These two functions, when used together, correspond to the Cascade Historian hist_gap_buffer dynamic library function.

Example

```
ngaps = HI_GapCountBuffer(n, values, gap);
n2 = n + ngaps;
new_values = ME_ZMalloc( n2 * sizeof (HI_stVALUE));
HI_GapFillBuffer (n, values, n2, new_values, gap);
```

HI_History

HI_History — creates a new history.

Syntax

```
#include <cogent.h>
ST_STATUS HI_History (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of the new history. It can be any valid Cascade DataHub point name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function creates a new history and assigns it a name. This is required before any other commands or functions on the specified history can be applied. The history will be enabled as soon as it is created. A history can be associated with a Cascade DataHub point, so that the history is updated whenever the datahub point is updated. (See the **register** command or the HI_Register function.)

Typically, the point and history names are the same, and the *point* argument need not be specified. By using the *point* argument, it becomes possible to associate more than one history to the same datahub point. In this case, a single datahub point update will cause all the associated histories to be updated. This can be useful if you need histories of the same data but with different deadband settings. (This could provide, for example, both a high-resolution history as well as a more compact history using the deadband filters). The *point* name can be completely different from the history name, or can be the same as one of the histories.

When multiple histories are associated with the same point, registering or unregistering one of the histories will do the same to all the other histories sharing the same point. This applies when defining a new history: if a point is associated with a history that has already registered, then the new history is also automatically registered. If control is required over which histories (that share a common point) receive data, then the **enable**/**disable** commands (or their corresponding functions) must be used.

If the history already exists, this function does nothing.

This function corresponds to the Cascade Historian **history** command and the `hist_history` dynamic library function.

HI_Interpolate

HI_Interpolate — queries history data for interpolation.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Interpolate (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname,
   char* interpolator,
   double start,
   double duration,
   int nxargs,
   char** xargs,
   int* bufferid
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of a history.

interpolator

The name of the interpolator to use, as described below.

start

The start of the time range of interest for query. If this value is 0, it defaults to the time of the first data value available for the history.

duration

The amount of time, in seconds, over which to perform the interpolation. If this value is 0, it defaults to the length of time between that specified by *start* and the time of the last data value available for the history.

nxargs

The number of extra arguments to the interpolator.

xargs

The array of extra arguments to the interpolator, as strings.

bufferid

A query ID number for the interpolation. This is a return argument.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function performs a query on history data, placing the result in a buffer for subsequent transfer. It generates a unique, sequentially numbered ID 'handle' for each query that is made, so clients can access the resulting buffer. The IDs have no meaning outside the Cascade Historian, and are only valid after a query and until the buffer is freed with the bufferIdDestroy command, the HIBufferIDDestroy Cogent C API function, or the hist_buffer_id_destroy dynamic library function, as appropriate.

This is the first of four steps (which can be done as commands, dynamic libary functions, or API functions) required to make an interpolation:

1. A call to **interpolate**, hist_interpolate, or HI_Interpolate performs a query on history data, placing the result in a buffer for subsequent transfer.
2. A call to **bufferIdLength**, hist_buffer_id_length, or HI_BufferIDLength gets the length of the interpolation buffer.
3. A call to **bufferIdData**, **bufferIdDataAscii**, hist_buffer_id_read, or HI_BufferIDRead brings in the data from the interpolation buffer.
4. A call to **bufferIdDestroy**, hist_buffer_id_destroy, or HI_BufferIDDestroy destroys the interpolation buffer. This should always be done to free up memory.

This task has been divided into four steps for convenience and performance. A buffer, once created, remains available to be read as required, potentially by multiple users, until it is no longer needed and can be destroyed. The buffers can also be very large, exceeding typical IPC message sizes, and requiring a relatively long time to transfer. That problem can be addressed by transferring the data in predictably-sized portions, allowing other processing to be done in between. The multi-step process defined above provides the flexibility needed for just such tailoring to the user's requirements. To run all four steps with one function call, see HI_InterpolateData in the Cogent C API manual or the hist_interpolate_data dynamic library function.

The following choices are available for the *interpolator* argument:

The NoInterpolator function is currently the only one that requires no additional parameters, making it possible to not supply *start* or *duration*. For the other interpolator functions, setting *start* or *duration* to 0 forces the default values.

- **NoInterpolator** simply returns all data that falls within the specified time range. No other processing is performed. When calling the HI_Interpolate function, this interpolator requires no extra arguments (*nxargs* = 0, *xargs* = NULL).
- **PeriodicInterpolator** generates data as Y vs. time on an even time interval. The first extra argument is a double-precision float indicating the time interval in seconds. The second (optional) extra argument is a double-precision float indicating the maximum gap time: if provided and the time between two data samples exceeds this threshold, then data cannot be interpolated between the points. This suppresses generating interpolated data during 'gaps' in the data.
- **RelativeInterpolator** generates Y vs. X at all known values of X. The first extra argument is a string indicating the history name for X. The *history* argument provided to **interpolate** command (or the *histname* argument provided to HI_Interpolate function) is used as the Y history.

- **`FixedRelativeInterpolator`** generates Y vs. X on an even time interval. The first extra argument is a string indicating the history name for X. The second extra argument is a double specifying the time interval. The *`history`* argument provided to **interpolate** command (or the *`histname`* argument provided to `HI_Interpolate` function) is used as the Y history.

This function corresponds to the Cascade Historian **interpolate** command and the `hist_interpolate` dynamic library function.

Example

The following example is the code for the HI_InterpolateData function. It demonstrates how HI_Interpolate is used with HI_BufferIDLength, HI_BufferIDRead, and HI_BufferIDDestroy.

```
ST_STATUS HI_InterpolateData (IP_hTASK historian, char* histname,
                              char* interpolator,
                              double start, double duration,
                              int nxargs, char** xargs,
                              HI_stVALUE** values, int *nvalues)
{
  static char           retbuf[256];
  ST_STATUS             status;
  int                   n, bufid;
  HI_stVALUE            *hvalues;

  status = HI_Interpolate (historian, retbuf, sizeof(retbuf),
                           histname, interpolator, start, duration,
                           nxargs, xargs, &bufid);
  if (status == ST_OK)
    {
      status = HI_BufferIDLength (historian, NULL, 0, bufid, &n);
      if (status == ST_OK)
        {
          hvalues = (HI_stVALUE*) ME_ZMalloc (n * sizeof(HI_stVALUE));
          if (hvalues)
            {
              status = HI_BufferIDRead (historian, NULL, 0,
                                        bufid, 0, n, hvalues);
              if (status == ST_OK)
                {
                  *values = hvalues;
                  *nvalues = n;
                }
              else
                {
                  ME_Free (hvalues);
                  *values = NULL;
                }
            }
        }
      HI_BufferIDDestroy (historian, NULL, 0, bufid);
    }
  return (status);
```

HI_InterpolateData

HI_InterpolateData — interpolates a history and returns data.

Syntax

```
#include <cogent.h>
ST_STATUS HI_InterpolateData (
   IP_hTASK historian,
   char* histname,
   char* interpolator,
   double start,
   double duration,
   int nxargs,
   char** xargs,
   HI_stVALUE** values,
   int* nvalues
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

histname

The name of a history.

interpolator

The name of the interpolator to use, as described below.

start

The start of the time range of interest for query. If this value is 0, it defaults to the time of the first data value available for the history.

duration

The amount of time, in seconds, over which to perform the interpolation. If this value is 0, it defaults to the length of time between that specified by *start* and the time of the last data value available for the history.

nxargs

The number of extra arguments to the interpolator.

xargs

The array of extra arguments to the interpolator, as strings.

values

A pointer to an HI_stVALUE*. This is a return argument.

nvalues

The number of elements in values. This is a return argument.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This is a convenience function that combines HI_Interpolate, HI_BufferIDLength, HI_BufferIDRead, and HI_BufferIDDestroy. It causes the Cascade Historian to perform an interpolation on its history data, producing a new, temporary, un-named history, which is then copied to the client program. The resulting data is returned in *values*, and is *nvalues* in length.

The code for this function can be viewed in the Example for HI_Interpolate.

You are responsible for freeing the *values*, using the function ME_Free. The xargs array is an array of NULL-terminated strings. Numeric arguments are passed as strings containing their printable ASCII representations.

The following choices are available for the *interpolator* argument:

The NoInterpolator function is currently the only one that requires no additional parameters, making it possible to not supply *start* or *duration*. For the other interpolator functions, setting *start* or *duration* to 0 forces the default values.

- **NoInterpolator** simply returns all data that falls within the specified time range. No other processing is performed. When calling the HI_Interpolate function, this interpolator requires no extra arguments (*nxargs* = 0, *xargs* = NULL).
- **PeriodicInterpolator** generates data as Y vs. time on an even time interval. The first extra argument is a double-precision float indicating the time interval in seconds. The second (optional) extra argument is a double-precision float indicating the maximum gap time: if provided and the time between two data samples exceeds this threshold, then data cannot be interpolated between the points. This suppresses generating interpolated data during 'gaps' in the data.
- **RelativeInterpolator** generates Y vs. X at all known values of X. The first extra argument is a string indicating the history name for X. The *history* argument provided to **interpolate** command (or the *histname* argument provided to HI_Interpolate function) is used as the Y history.
- **FixedRelativeInterpolator** generates Y vs. X on an even time interval. The first extra argument is a string indicating the history name for X. The second extra argument is a double specifying the time interval. The *history* argument provided to **interpolate** command (or the *histname* argument provided to HI_Interpolate function) is used as the Y history.

This function corresponds to the Cascade Historian hist_interpolate_data dynamic library function.

HI_Latest

HI_Latest — gives the latest value in a history.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Latest (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname,
   HI_stVALUE* data
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of a history.

data

The value and time of the last data item in the history.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function returns the value for the last data item in the history. This item may be either in a file or in memory.

This function corresponds to the Cascade Historian **latest** command and the hist_latest dynamic library function.

HI_Length

HI_Length — finds the total number of values in a history.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Length (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histname,
   int* length
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histname

The name of a history.

length

The return value containing the number of values in the history.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function queries the history for the total number of values available. It returns the result in *length*.

This function corresponds to the Cascade Historian **length** command and the hist_length dynamic library function.

HI_List

HI_List — finds the name of available histories.

Syntax

```
#include <cogent.h>
ST_STATUS HI_List (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histpattern,
   int offset,
   int maxcount,
   char** histories,
   int* count
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

A required buffer containing an error message.

buflen

The length in bytes of *retbuf*. This must be a valid non-zero length.

histpattern

A globbing pattern specifying a group of histories.

offset

An integer specifying a starting point in the list generated by *histpattern*.

maxcount

The maximum number of history names to be returned.

histories

An array of char* at least *maxcount* long, to contain up to min(*count*,*maxcount*) history names.

count

The return value containing the number of histories (same as HI_Count).

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* will contain the list of history names, parsed into NULL-terminated string, and pointed to by the *histories* array (see the note in Description).

Description

This function queries the Cascade Historian for the names of the histories currently being kept that match the *histpattern*. You must supply an array to contain the pointers to the name strings.

The history names are returned as strings in *retbuf*, and the pointers returned in *histories* point to the parsed contents of *retbuf*. To preserve the history names, a reserved buffer should be used, or no further historian API calls should be made until the strings are no longer needed.

In cases where the number of histories, or the length of their names, is such that the combination exceeds *buflen* or the length of the inter-process communication buffer, not all histories will be listed, and the *count* argument will be less than the value obtained from `HI_Count`. In these cases, the *offset* parameter can be used to receive a portion of the list. Starting *offset* at `0`, `count` can then be used to increment `offset`, enabling arbitrarily long lists of names to be obtained without concern for the length of *retbuf*, the IPC buffer, or the size of the *histories* array (see Example).

This function corresponds to the Cascade Historian **list** command and the `hist_list` dynamic library function.

Example

```
void ListAllHistories (IP_hTASK historian)
{
     ST_STATUS status;
     char      *histpattern = "*";
     char      *histories[100];
     int       nHist, start, count;
     /* assume retbuf, buflen are globals */

 status = HI_Count(historian, retbuf, buflen, histpattern, &nHist);
 /* loop querying for names until all have been received */
 for (start=0; start < nHist && status == ST_OK; start += count)
      {
  status = HI_List(historian, retbuf, buflen, histpattern,
                                    100, histories, &count);
            /* process the subset of names */
            for (i=0; i<count; i++)
                printf ("%s\n", histories[i]);
      }
}
```

HI_ScaleBuffer

HI_ScaleBuffer — performs offset and scaling operations.

Syntax

```
#include <cogent.h>
ST_STATUS HI_ScaleBuffer (
   int nvalues,
   HI_stVALUE* values,
   double xscale,
   double xoff,
   double yscale,
   double yoff
);
```

Arguments

nvalues

The number of values in the *values* array.

values

An array of HI_stVALUE structures.

xscale, xoff

The X value of the data pairs is transformed as follows:

```
x' = x * xscale + xoffset
```

yscale, yoff

The Y value of the data pairs is transformed as follows:

```
y' = y * yscale + yoffset
```

Returns

t (true) if the transformation was performed with no errors, otherwise nil.

Description

This function performs an offset and scaling transformation on a binary buffer of X-Y data pairs. The buffer is modified in place (i.e., a new buffer is not created, and the old values cannot be recovered). This function can be used to adjust data for graphing.

This function corresponds to the Cascade Historian hist_scale_buffer dynamic library function.

HI_StatBuffer

HI_StatBuffer — generates summary statistics on Y-values.

Syntax

```
#include <cogent.h>
ST_STATUS HI_StatBuffer (
   int nvalues,
   HI_stVALUE* values,
   double* minptr,
   double* maxptr,
   double* meanptr,
   double* stddevptr
);
```

Arguments

nvalues

The number of values in the *values* array.

values

An array of HI_stVALUE structures.

minptr

A pointer to the minimum Y-value.

maxptr

A pointer to the maxium Y-value.

meanptr

A pointer to the mean Y-value.

stddevptr

A pointer to the standard deviation of the Y-values.

Returns

t (true) if the function completed successfully, otherwise nil.

Description

This function generates a set of summary statistics on the Y-value data in a buffer: minimum value, maximum value, mean, and standard deviation. The X-value data is ignored. The buffer data would typically represent the result of a query for all values of a specific history (previously retrieved with the Cascade Historian hist_read_buffer function). The summary statistics can then be used to provide some idea of the nature of the data, or could be used to automatically scale the display of the data.

This function corresponds to the Cascade Historian hist_stat_buffer dynamic library function.

HI_Register

HI_Register — registers histories with the Cascade DataHub.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Register (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histpattern
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histpattern

A globbing pattern specifying a group of histories.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function causes the Cascade Historian to register each history matching the pattern with the Cascade DataHub point of the same name. Any subsequent value changes transmitted by the Cascade DataHub will be treated exactly the same as if **add** or HI_Add were called with that history, value and timestamp. If a history has already been registered, this function does nothing. If the Cascade DataHub is not running, no registration occurs. If multiple histories are associated with the same point, registering or unregistering one of the histories will do the same to all the other histories sharing the same point.

A Cascade DataHub point is a unit of storage in the datahub. Points have a name (a string), a value (real, integer, or string), and a time associated with them. They can be written to the Cascade DataHub in a number of ways, including for example the Cascade DataHub **writept** command and the Gamma write_point function. The Cascade Historian receives a point's data from the Cascade DataHub by registering for it with the **register** command or HI_Register function.

This function corresponds to the Cascade Historian **register** command and the hist_register dynamic library function.

HI_Unregister

HI_Unregister — unregisters histories with the Cascade DataHub.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Unregister (
   IP_hTASK historian,
   char* retbuf,
   int buflen,
   char* histpattern
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

An optional buffer containing an error message.

buflen

The length in bytes of *retbuf*. If *retbuf* is non-NULL, this must be a valid non-zero length. If *retbuf* is NULL, this parameter is ignored.

histpattern

A globbing pattern specifying a group of histories.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function causes the Cascade Historian to unregister each history matching the pattern from the Cascade DataHub. The history will cease to receive exceptions when the point value data changes. If multiple histories are associated with the same point, registering or unregistering one of the histories will do the same to all the other histories sharing the same point.

If the history is not currently registered with the Cascade DataHub, this function does nothing.

This function corresponds to the Cascade Historian **unregister** command and the hist_unregister dynamic library function.

HI_Version

HI_Version — gets the version number of the Cascade Historian.

Syntax

```
#include <cogent.h>
ST_STATUS HI_Version (
   IP_hTASK historian,
   char* retbuf,
   int buflen
);
```

Arguments

historian

The task pointer to the Cascade Historian program.

retbuf

A *required* buffer containing an error message.

buflen

The length in bytes of *retbuf*. This must be a valid non-zero length.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a NULL-terminated character string with an error message. If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function returns the version number of the Cascade Historian in *retbuf*. The *retbuf* and *buflen* parameters are not optional in this case. *buflen* must be at least 32 characters.

This function corresponds to the Cascade Historian **version** command and the hist_version dynamic library function.

VI. Inter-Process Communication Functions

Table of Contents

IP_AddFDHandler 162
IP_AttachPhoton 163
IP_AttachPhotonMainloop 164
IP_ConnectToPort 165
IP_ConnectToService 166
IP_DetachPhotonMainloop 167
IP_GetChannelID 168
IP_GetConnectionID 169
IP_IsPulse 170
IP_ListenToPort 171
IP_ListenToService 172
IP_MsgCascade 173
IP_MsgCreate 174
IP_MsgData 175
IP_MsgDefaultSize 176
IP_MsgDestroy 177
IP_MsgInfoReply 178
IP_MsgInfoReplyRaw 179
IP_MsgLisp 180
IP_MsgRaw 181
IP_MsgRawData 182
IP_MsgResize 183
IP_NserveAdd 184
IP_NserveClose 185
IP_NserveInit 186
IP_NserveInitMyself 187
IP_NserveLookup 188
IP_NserveLookupId 189
IP_NserveLookupName 190
IP_NservePackTaskInfo 191
IP_NserveQueryNameCount 193
IP_NserveQueryNames 194
IP_NserveReattach 195
IP_NserveRemove 196
IP_NserveSetDomain 197
IP_pfTaskComp 198
IP_PhotonGUIFilter 199

IP_PhotonGUIHandler 200
IP_ProcessMessage 201
IP_PulseDestroy 202
IP_PulseNew 203
IP_PulseTimed 204
IP_PulseTrigger 205
IP_QueueClose 206
IP_QueueOpen 207
IP_QueueRead 208
IP_QueueStrerror 209
IP_QueueWait 210
IP_QueueWrite 212
IP_Receive 213
IP_ReceiveNonblock 215
IP_RemoveFDHandler 216
IP_Reply 217
IP_ReplyRaw 218
IP_SelectFD 219
IP_SetChannelID 220
IP_SetConnectionID 221
IP_SetGUIHandler 222
IP_TaskCloseAsync 223
IP_TaskCloseSync 224
IP_TaskConnect 225
IP_TaskCopy 226
IP_TaskCreate 227
IP_TaskCreateMe 228
IP_TaskDefaultDomain 229
IP_TaskDestroy 230
IP_TaskFindID 231
IP_TaskFindName 232
IP_TaskInitAsync 233
IP_TaskInitAsyncWrites 234
IP_TaskIntern 235
IP_TaskNew 236
IP_TaskSendAsync 237
IP_TaskSendRaw 238
IP_TaskSendSync 239
IP_TaskSetDomain 240
IP_TaskSetInfo 241
IP_TaskSetQname 243
IP_TaskUnintern 244
IP_TaskWaitAsync 245

IP_TaskZero 246
IP_TimerTime 247
IP_UnselectFD 248

IP_AddFDHandler

`IP_AddFDHandler` — tells `IP_Receive` to accept file input.

Syntax

```
#include <cogent.h>
int IP_AddFDHandler (
   int fd
   );
```

Arguments

`fd`

An open file descriptor.

Returns

`-1` on failure and `errno` is set, or an arbitrary number other than `-1` on success.

Description

This function tells `IP_Receive` that it should accept input from the given file descriptor, and return `IP_FD` whenever a read event occurs on that descriptor.

See Also

Receiving Messages and Events, `IP_RemoveFDHandler`

IP_AttachPhoton

IP_AttachPhoton — not fully documented.

Syntax

```
#include <cogent.h>
int IP_AttachPhoton (
   void
   );
```

Arguments

Not yet documented.

Returns

Not yet documented.

Description

Not yet documented.

See Also

Photon Functions, IP_AttachPhotonMainloop

IP_AttachPhotonMainloop

IP_AttachPhotonMainloop — not fully documented.

Syntax

```
#include <cogent.h>
int IP_AttachPhotonMainloop (
   IP_pfMessageHandler handler
   );
```

Arguments

Not yet documented.

Returns

Not yet documented.

Description

Not yet documented.

See Also

Photon Functions, IP_DetachPhotonMainloop

IP_ConnectToPort

`IP_ConnectToPort` — resolves a host name and connects to a port.

Syntax

```
#include <cogent.h>
int IP_ConnectToPort (
   char* host,
   int port
);
```

Arguments

`host`

A host name.

`port`

A port number.

Returns

A file descriptor (socket) on success, or `-1` on failure and `errno` is set.

Description

This function resolves the host name, which may be either a fully qualified host name or an IP address as an ASCII string containing a dotted quad. It then attempts to connect to the specified port on that host.

See Also

Working with TCP/IP, `IP_ListenToService`, `IP_ConnectToService`

IP_ConnectToService

`IP_ConnectToService` — like `IP_ConnectToPort`, but uses the service name.

Syntax

```
#include <cogent.h>
int IP_ConnectToService (
   char* host,
   char* service
);
```

Arguments

`host`

A host name.

`service`

The service name, as found in `/etc/services`.

Returns

A file descriptor (socket) on success, or -1 on failure and `errno` is set.

Description

This function looks up the service name, resolving it to a port number before calling `IP_ConnectToPort`.

See Also

Working with TCP/IP, `IP_ListenToService`, `IP_ConnectToPort`

IP_DetachPhotonMainloop

IP_DetachPhotonMainloop — not fully documented.

Syntax

```
#include <cogent.h>
int IP_DetachPhotonMainloop (
   void
   );
```

Arguments

Not yet documented.

Returns

Not yet documented.

Description

Not yet documented.

See Also

Photon Functions, IP_AttachPhotonMainloop

IP_GetChannelID

`IP_GetChannelID` — returns the channel ID for IP library use.

Syntax

```
#include <cogent.h>
int IP_GetChannelID (
   void
   );
```

Arguments

None.

Returns

The channel ID being used by the IP library, or `-1` if no channel could be created.

Description

This function returns the channel ID to be used by the IP library functions. If there is no current channel set, then this function creates a new channel and returns that. Subsequent calls to this function will return the same channel.

See Also

Connections and Channels, `IP_SetChannelID`

IP_GetConnectionID

`IP_GetConnectionID` — returns the connection ID for IP library use.

Syntax

```
#include <cogent.h>
int IP_GetConnectionID (
   void
   );
```

Arguments

None.

Returns

The connection ID being used by the IP library, or `-1` if no connection could be created.

Description

This function returns the connection ID to be used by the IP library functions. If there is no current connection, then this function creates a new connection and returns that. This function calls `IP_GetChannelID`, possibly also creating a new channel.

See Also

Connections and Channels, `IP_SetConnectionID`

IP_IsPulse

IP_IsPulse — validates a received message against a pulse ID.

Syntax

```
#include <cogent.h>
int IP_IsPulse (
   int pulse,
   int rcvid,
   void* msg
);
```

Arguments

pulse

A pulse ID as generated by IP_PulseNew.

rcvid

The rcvid field from an IP_MsgInfo structure.

msg

The pointer to a message received through IP_Receive.

Returns

0 if the *pulse* does not correspond to the *rcvid* and *msg*, or non-zero if the *rcvid* and *msg* do represent the *pulse*.

Description

This function compares the *rcvid* and *msg* to the *pulse*, and determines whether they really represent an occurrence of the *pulse*.

See Also

Pulses and Timers

IP_ListenToPort

IP_ListenToPort — is a wrapper for listen.

Syntax

```
#include <cogent.h>
int IP_ListenToPort (
   int port,
   int backlog
);
```

Arguments

port

A port number.

backlog

The number of connection requests that are allowed to be simultaneously pending.

Returns

The file descriptor (socket), or -1 and errno is set.

Description

This function wraps the listen library call, and registers the resulting file descriptor with the IP_Receive function.

See Also

Working with TCP/IP, IP_ListenToService, IP_ConnectToPort

IP_ListenToService

`IP_ListenToService` — like `IP_ListenToPort`, but uses the service name.

Syntax

```
#include <cogent.h>
int IP_ListenToService (
   char* service,
   int backlog
);
```

Arguments

`service`

The service name, as found in `/etc/services`.

`backlog`

The number of connection requests that are allowed to be simultaneously pending.

Returns

The file descriptor (socket), or -1 and `errno` is set.

Description

This function wraps the `listen` library call, and registers the resulting file descriptor with the `IP_Receive` function. Prior to calling `listen`, this function looks up the service name to produce a port number which is given to `IP_ListenToPort`.

See Also

Working with TCP/IP, `IP_ListenToPort`, `IP_ConnectToService`

IP_MsgCascade

IP_MsgCascade — writes message data to an IP_MsgBuffer.

Syntax

```
#include <cogent.h>
int IP_MsgCascade (
   IP_Msg* message,
   void* data,
   int len,
   int subtype,
   int status
);
```

Arguments

message

A pointer to a message.

data

Data to copy into message->msg->data.

len

The amount of data to copy.

subtype

The message subtype to be set.

status

The message status to be set.

Returns

0 on success, or -1 if the message could not be resized, and errno is set:

- ENOSYS - the message is either not resizable or not dynamic.
- ENOMEM - the memory reallocation returned NULL.

The msg portion of message is set to NULL.

Description

This function assumes that the message points to an IP_MsgBuffer, and writes *data* into its data portion for *len* bytes. If *len* is greater than the currently allocated message length, the message is resized. If the *data* argument points to the same location as the message's data, then no copy is performed. The message's length, status and subtype are all updated regardless. The message type is set to IP_MSG_COGENT.

See Also

Messages, IP_MsgResize

IP_MsgCreate

IP_MsgCreate — creates an IP_Msg structure.

Syntax

```
#include <cogent.h>
IP_Msg* IP_MsgCreate (
   void* data,
   intdatalen,
   intnoresize
);
```

Arguments

data

The initial data in the message, or NULL.

datalen

The length of the allocated data buffer.

noresize

If non-zero, indicates that *datalen* is fixed.

Returns

A pointer to a new IP_Msg.

Description

This function creates a new IP_Msg structure whose message is an IP_MsgBuffer with a data buffer that is *datalen* bytes in length.

If *data* is non-NULL, then *datalen* bytes are copied into the newly allocated data buffer in the IP_MsgBuffer component of the IP_Msg. If *noresize* is 0, then the IP library may enlarge the data buffer in the future to accomodate requests larger than *datalen*. If *datalen* is 0, then IP_MsgDefaultSize is used. The message is marked as dynamic.

See Also

Messages, IP_MsgDestroy

IP_MsgData

IP_MsgData — returns a pointer to the data payload of an IP_Msg structure.

Syntax

```
#include <cogent.h>
char* IP_MsgData (
   IP_Msg* msg
   );
```

Arguments

msg

A pointer to an IP_Msg structure.

Returns

A pointer to the data portion of a message.

Description

This function returns the data portion of an IP_Msg structure. This is the buffer of characters to be transmitted, excluding any message header information.

IP_MsgData(*message*) is equivalent to message->msg->data.

See Also

Messages

IP_MsgDefaultSize

IP_MsgDefaultSize — gets the default size of interprocess messages.

Syntax

```
#include <cogent.h>
int IP_MsgDefaultSize (
   void
   );
```

Arguments

None.

Returns

The value of the environment variable IP_MSG_DEFAULT_SIZE if it is set, or 4096. If IP_MSG_DEFAULT_SIZE <= 0 then 4096 is returned. If IP_MSG_DEFAULT_SIZE > 65400, then 65400 is returned.

Description

This function returns the default message size for interprocess messages. The messaging primitives (Send/Receive/Reply) do not easily provide a sender information about the amount of space allocated by the receiver. By using the IP_MsgDefaultSize, all processes can agree on the message sizes being transmitted. The default size normally refers to the number of bytes in the data portion of the IP_MsgBuffer pointed to by an IP_Msg.

See Also

Messages, IP_MsgResize

IP_MsgDestroy

IP_MsgDestroy — frees memory associated with a message.

Syntax

```
#include <cogent.h>
void IP_MsgDestroy (
   IP_Msg* message
   );
```

Arguments

message

A pointer to a message.

Returns

Nothing.

Description

This function cleans up all memory associated with the *message*. If the *message* is marked as dynamic, then the msg portion of the IP_Msg is also freed.

See Also

Messages, IP_MsgCreate

IP_MsgInfoReply

IP_MsgInfoReply — replies to an IP_SYNC message using IP_MsgInfo.

Syntax

```
#include <cogent.h>
int IP_MsgInfoReply (
   IP_MsgInfo* orig,
   IP_Msg* rmsg
);
```

Arguments

orig

The original msginfo as filled by IP_Receive.

rmsg

The reply message.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function replies to a previously received IP_SYNC message by using the information in the IP_MsgInfo structure filled by IP_Receive. The *rmsg* contains the information to be transmitted as the reply.

See Also

Replying to Messages, IP_Reply

IP_MsgInfoReplyRaw

IP_MsgInfoReplyRaw — replies to an IP_RAW message using IP_MsgInfo.

Syntax

```
#include <cogent.h>
int IP_MsgInfoReplyRaw (
   IP_MsgInfo* orig,
   char* msg,
   int len
);
```

Arguments

orig

The original msginfo as filled by IP_Receive.

msg

The message buffer to be transmitted.

len

The length of the message buffer.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function replies to a previously received IP_RAW message by using the information in the IP_MsgInfo structure filled by IP_Receive. *len* bytes of *msg* are transmitted as the reply message.

See Also

Replying to Messages, IP_ReplyRaw

IP_MsgLisp

IP_MsgLisp — constructs a formatted text message.

Syntax

```
#include <cogent.h>
int IP_MsgLisp (
   IP_Msg* message,
   int status,
   char* format ...
);
```

Arguments

message

A pointer to a message.

status

The message status to be set.

format

A UT_LispString style format string followed by a variable number of arguments.

Returns

0 on success; or -1 if the message could not be resized, and errno is set:

- ENOSYS - the message is either not resizable or not dynamic.
- ENOMEM - the memory reallocation returned NULL.

The msg portion of message is set to NULL.

Description

This function constructs a formatted text message in the data portion of the message by calling UT_LispString. If the message data is too short to hold the formatted text, then the message size is increased by progressive factors of two and the format attempt is repeated until the message can be written or until a call to IP_MsgResize fails. The resulting message is submitted to IP_MsgCascade with a subtype of IP_SUB_LISP and the status as supplied in the status argument to IP_MsgLisp.

See Also

Messages, IP_MsgResize

IP_MsgRaw

IP_MsgRaw — like IP_MsgCascade, with IP_SUB_RAW for its *subtype*.

Syntax

```
#include <cogent.h>
int IP_MsgRaw (
   IP_Msg* message,
   void* data,
   int len
);
```

Arguments

message

A pointer to a message.

data

Data to copy into message->msg->data.

len

The amount of data to copy.

Returns

0 on success; or -1 if the message could not be resized, and errno is set:

- ENOSYS - the message is either not resizable or not dynamic.
- ENOMEM - the memory reallocation returned NULL.

The msg portion of message is set to NULL.

Description

This function is equivalent to: IP_MsgCascade (*message*, *data*, *len*, IP_SUB_RAW, 0)

See Also

Messages, IP_MsgCascade

IP_MsgRawData

IP_MsgRawData — gives a pointer to IP_RAW message data.

Syntax

```
#include <cogent.h>
IP_MsgRawData (
   msg
   );
```

Arguments

msg

A pointer to an IP_Msg structure.

Returns

A void pointer to the raw message data, defined as:

```
((void*)((msg)->msg))
```

Description

This is a macro that finds and returns the correct pointer to message data—if and only if the message type (as returned by IP_Receive) is IP_RAW.

See Also

Messages

IP_MsgResize

IP_MsgResize — resizes the IP_MsgBuffer, if possible.

Syntax

```
#include <cogent.h>
int IP_MsgResize (
   IP_Msg* message,
   int datalen
);
```

Arguments

message

A pointer to a message.

datalen

The new length of the data portion of the IP_MsgBuffer, pointed to as message->msg.

Returns

0 on success, -1 on failure and errno is set:

- ENOSYS - the message is either not resizable or not dynamic.
- ENOMEM - the memory reallocation returned NULL.

The msg portion of message is set to NULL.

Description

This function assumes that the message encapsulates an IP_MsgBuffer, and resizes the data portion of that IP_MsgBuffer to *datalen* bytes through a call to ME_Realloc.

See Also

Messages, IP_MsgDefaultSize

IP_NserveAdd

IP_NserveAdd — adds an entry to **nserve**.

Syntax

```
#include <cogent.h>
int IP_NserveAdd (
   char* name,
   char* domain,
   char* qname,
   int nid,
   int pid,
   int chid
);
```

Arguments

name

A task name.

domain

A domain name, or NULL.

qname

A queue name, or NULL.

nid

A node ID.

pid

A process ID.

chid

A channel ID.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function adds an entry to **nserve**, associating the *name* with *domain*, *qname*, *nid*, *pid* and *chid*.

If *domain* is NULL, then the default domain is used. If *qname* is NULL, then a blank queue name is used. The actual nid/pid/chid combination need not actually exist. The resulting **nserve** entry will be removed when the process that created it exits. A single process may add any number of names.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, IP_NserveRemove

IP_NserveClose

IP_NserveClose — closes the channel to **nserve**.

Syntax

```
#include <cogent.h>
int IP_NserveClose (
   IP_Task* task
   );
```

Arguments

task

A task structure referring to **nserve**.

Returns

0

Description

This function closes the channel to **nserve**, which produces exactly the same result as if the process had died. **nserve** will generate a task death notice for each of the names associated with this process.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, IP_NserveReattach

IP_NserveInit

`IP_NserveInit` — creates a task structure and informs **nserve**.

Syntax

```
#include <cogent.h>
IP_Task* IP_NserveInit (
   char* name,
   char* domain,
   char* qname,
   int qsize,
   int flags
);
```

Arguments

name

A public name for this task. This must be unique system-wide.

domain

A Cascade DataHub domain name, or `NULL`.

qname

A queue name, or `NULL`.

qsize

The number of messages in the queue, or `0`.

flags

A bitwise-`OR` of initialization flags, or `0`.

Returns

A task structure pointer on success, or `NULL` on failure, and `errno` is set.

Description

This function creates a task structure for the current task, and sends the necessary information to **nserve**. If the *domain* is `NULL`, the default domain is used. If the *qname* is null, then the task will have no input queue. If the *qsize* is `0`, then it will be assigned a default value (currently `100`). There are currently no flags.

Task names must be unique system-wide. This restriction is enforced as much as possible by **nserve**, but there are certain unavoidable situations that may allow multiple processes on different nodes to register the same name. Some care should be taken by the programmer to ensure that this does not happen.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, `IP_NserveInitMyself`

IP_NserveInitMyself

IP_NserveInitMyself — declares current task structure information to **nserve**.

Syntax

```
#include <cogent.h>
int IP_NserveInitMyself (
   IP_Task* myself,
   int flags
);
```

Arguments

myself

The task structure for the current task.

flags

A bitwise-OR of initialization flags, or 0.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function declares the information stored in the task structure to **nserve**, through a call to IP_NserveInit. There are currently no flags.

See Also

Cascade NameServer Functions, **nserve**, IP_NserveInit

IP_NserveLookup

IP_NserveLookup — fills in a known task structure.

Syntax

```
#include <cogent.h>
int IP_NserveLookup (
   char* name,
   IP_Task* task
);
```

Arguments

name

A task name.

task

An empty task structure to hold the result.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function queries **nserve** for the first task corresponding to *name*. The task structure is filled in with all information from **nserve**. Strings are allocated as necessary, and will be deallocated when the task pointer is submitted to IP_TaskDestroy.

IP_NserveLookup assumes that the memory allocated for the task structure is uninitialized. After you have called IP_NserveLookup once on a given task structure, each subsequent time you call it on the same task structure, new buffers are allocated for names, but the old pointers are not freed. To eliminate this potential memory leak, you can use this sequence:

```
task = IP_TaskNew();
IP_NserveLookup (name, task);
IP_TaskDestroy (task);
```

or this one:

```
task = IP_NserveLookupName (name);
if (task) IP_TaskDestroy (task);
```

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, IP_NserveLookupID, IP_NserveLookupName

IP_NserveLookupId

`IP_NserveLookupId` — finds a task by node, process and channel ID.

Syntax

```
#include <cogent.h>
int IP_NserveLookupId (
   int nid,
   int pid,
   int chid,
   IP_Task* task
);
```

Arguments

`nid`

A node ID.

`pid`

A process ID.

`chid`

A channel ID.

`task`

An empty task structure to hold the result.

Returns

`0` on success, or `-1` on failure and `errno` is set.

Description

This function queries **nserve** for the first task whose `nid`, `pid` and `chid` exactly match the provided values. The task structure is filled in with all information from **nserve**. Strings are allocated as necessary, and will be deallocated when the task pointer is submitted to `IP_TaskDestroy`.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, `IP_NserveLookup`, `IP_NserveLookupName`

IP_NserveLookupName

IP_NserveLookupName — allocates and fills in a new task structure.

Syntax

```
#include <cogent.h>
IP_Task* IP_NserveLookupName (
   char* name
   );
```

Arguments

name

A task name.

Returns

A pointer to a task structure on success, or NULL on failure and errno is set.

Description

This function is a convenience wrapper on IP_NserveLookup that allocates a new IP_Task structure and then attempts to fill it. If the *name* is not registered with **nserve**, the new task structure is destroyed.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, IP_NserveLookup, IP_NserveLookupID

IP_NservePackTaskInfo

IP_NservePackTaskInfo — makes a Lisp-parseable version of task information.

Syntax

```
#include <cogent.h>
int IP_NservePackTaskInfo (
   char* buffer,
   int maxlen,
   char* prefix,
   char* name,
   char* domain,
   char* qname,
   int nid,
   int pid,
   int chid
);
```

Arguments

buffer

A buffer used to hold the packed information.

maxlen

The maximum length of buffer.

prefix

A string to be prepended to the packed information.

name

The task name.

domain

The task's domain name, or NULL.

qname

The task's queue name, or NULL.

nid

The task's node ID.

pid

The task's process ID.

chid

The task's input channel ID.

Returns

0 on success. -1 on failure and errno is set:

- ENOMEM - the buffer size was too small to hold the result.

Description

This function creates a Lisp-parseable representation of the task information, suitable for transmission to **nserve** and associated library functions. If *domain* is NULL, the default domain is used. If *qname* is NULL, then a blank queue name is used. The *nid*, *pid* and *chid* parameters must all be specified.

See Also

The Cascade NameServer, Cascade NameServer Functions

IP_NserveQueryNameCount

`IP_NserveQueryNameCount` — gives the number of registered names.

Syntax

```
#include <cogent.h>
int IP_NserveQueryNameCount (
   void
   );
```

Arguments

None.

Returns

The number of names `>= 0` on success, or `-1` on failure and `errno` is set.

Description

This function queries **nserve** for the number of currently registered names.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, `IP_NserveQueryNames`

IP_NserveQueryNames

IP_NserveQueryNames — fills an array with **nserve**'s names.

Syntax

```
#include <cogent.h>
int IP_NserveQueryNames (
   char** names,
   int maxnames
);
```

Arguments

names

An array of pointers to be filled in.

maxnames

The number of entries in the names array.

Returns

The number of names actually received, or -1 on error and errno is set.

Description

This function queries **nserve** for up to *maxnames* names, allocates memory for each name, and adds it to the *names* array. It is the programmer's responsibility to call ME_Free on each pointer in the *names* array to clean up this memory.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, IP_NserveQueryNameCount

IP_NserveReattach

IP_NserveReattach — closes and renews all task connections and queues.

Syntax

```
#include <cogent.h>
int IP_NserveReattach (
   IP_Task* task
   );
```

Arguments

task

A valid task pointer.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function will close all connections and queues associated with the *task*, and will attempt to look up the *task* name again on **nserve**, replacing any new information that it acquires. This is useful if the pointer to the task structure is widely used in an application, and the programmer wishes to re-locate a dead task without having to visit all uses of the task pointer.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, IP_NserveClose

IP_NserveRemove

`IP_NserveRemove` — removes an entry from **nserve**.

Syntax

```
#include <cogent.h>
int IP_NserveRemove (
   char* name
   );
```

Arguments

name

A task name.

Returns

`0` on success, or `-1` on failure and `errno` is set.

Description

This function removes the *name* entry from **nserve**. No check is performed to ensure that the requesting task in fact "owns" the name being removed. This will trigger the same response by **nserve** as if the process referred to by *name* had exited, except that any other names associated with that process will remain.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**, `IP_NserveAdd`

IP_NserveSetDomain

`IP_NserveSetDomain` — changes a task's domain name.

Syntax

```
#include <cogent.h>
int IP_NserveSetDomain (
   IP_Task* task,
   char* domain
);
```

Arguments

`task`

A pointer to task structure for this task.

`domain`

A new domain name for this task.

Returns

`0` on success, or `-1` on failure and `errno` is set.

Description

This function modifies the domain name of the *`task`*, and informs **nserve** of that change.

See Also

The Cascade NameServer, Cascade NameServer Functions, **nserve**

IP_pfTaskComp

IP_pfTaskComp — compares two tasks for equality.

Syntax

```
#include <cogent.h>
int IP_pfTaskComp (
   const void* t1,
   const void* t2
);
```

Arguments

t1

A pointer to a task structure.

t2

A pointer to a task structure.

Returns

<0 if *t1* is ordinally less than *t2*, 0 if the two tasks are equal, and >0 if *t1* is ordinally greater than *t2*.

Description

This function compares two tasks for equality by comparing their nid, pid and chid values.

See Also

Task Structures

IP_PhotonGUIFilter

IP_PhotonGUIFilter — not fully documented.

Syntax

```
#include <cogent.h>
int IP_PhotonGUIFilter (
   int rcvid,
   void* msg,
   int length
);
```

Arguments

Not yet documented.

Returns

Not yet documented.

Description

Not yet documented.

See Also

Photon Functions, IP_PhotonGUIHandler

IP_PhotonGUIHandler

IP_PhotonGUIHandler — not fully documented.

Syntax

```
#include <cogent.h>
int IP_PhotonGUIHandler (
   void* msg,
   int length
);
```

Arguments

Not yet documented.

Returns

Not yet documented.

Description

Not yet documented.

See Also

Photon Functions, IP_PhotonGUIFilter

IP_ProcessMessage

IP_ProcessMessage — classifies messages for IP_Receive.

Syntax

```
#include <cogent.h>
int IP_ProcessMessage (
   IP_Task* myself,
   int rcvid,
   void* msgbuffer,
   int length,
   IP_MsgInfo* msginfo
);
```

Arguments

myself

A task structure referring to the current task.

rcvid

The rcvid from the most recently received message.

msgbuffer

The most recently received message.

length

The length of msgbuffer.

msginfo

A structure to be filled with message information.

Returns

The message type.

Description

This function is used by IP_Receive to classify messages. Do not call this function directly.

See Also

Receiving Messages and Events, IP_Receive

IP_PulseDestroy

IP_PulseDestroy — destroys a pulse.

Syntax

```
#include <cogent.h>
int IP_PulseDestroy (
   int pulse
   );
```

Arguments

pulse

A pulse ID as generated by IP_PulseNew.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function destroys a pulse, cleaning up any associated resources.

See Also

Pulses and Timers, IP_PulseNew, IP_PulseDestroy

IP_PulseNew

`IP_PulseNew` — creates a new pulse.

Syntax

```
#include <cogent.h>
int IP_PulseNew (
   void
   );
```

Arguments

None.

Returns

A pulse ID on success, or `-1` on failure and `errno` is set.

Description

This function creates a new pulse.

See Also

Pulses and Timers, `IP_PulseTrigger`, `IP_PulseDestroy`

IP_PulseTimed

IP_PulseTimed — sets up timers to trigger pulses.

Syntax

```
#include <cogent.h>
timer_t IP_PulseTimed (
   int pulse,
   time_t init_sec,
   time_t init_nsec,
   time_t interval_sec,
   time_t interval_nsec
);
```

Arguments

pulse

A pulse ID as generated by IP_PulseNew.

init_sec

Seconds of delay prior to the first timer expiry.

init_nsec

Nanoseconds of delay prior to the first timer expiry.

interval_sec

Seconds of delay for subsequent timing intervals.

interval_nsec

Nanoseconds of delay for subsequent timing intervals.

Returns

A timer identifier on success, or -1 on failure and errno is set.

Description

This function causes a pulse to be triggered after a given number of *init_sec + init_nsec*, and then to trigger on a regular interval every *interval_sec + interval_nsec* thereafter.

See Also

Pulses and Timers, IP_TimerTime

IP_PulseTrigger

`IP_PulseTrigger` — immediately sends a pulse.

Syntax

```
#include <cogent.h>
int IP_PulseTrigger (
   IP_Task* task,
   int pulse
);
```

Arguments

`task`

The task to which the pulse is sent.

`pulse`

The pulse ID.

Returns

`0` on success, or `-1` on failure and `errno` is set.

Description

This function causes the *`pulse`* to be sent immediately to the *`task`*. It is the programmer's responsibility to ensure that the *`task`* and *`pulse`* are associated.

See Also

Pulses and Timers, `IP_PulseNew`, `IP_PulseDestroy`

IP_QueueClose

`IP_QueueClose` — closes a queue.

Syntax

```
#include <cogent.h>
int IP_QueueClose (
   int qid
   );
```

Arguments

`qid`

A queue ID.

Returns

`0` on success, or `-1` on failure and `errno` is set.

Description

This function closes the queue specified by *`qid`*. If this process is the creator of the queue, then the queue is deleted.

See Also

Cascade QueueServer Functions, `IP_QueueOpen`

IP_QueueOpen

IP_QueueOpen — opens a queue for reading or writing.

Syntax

```
#include <cogent.h>
int IP_QueueOpen (
   char* name,
   int nid,
   int max_msgs,
   int mode
);
```

Arguments

name

A queue name.

nid

A node ID on which to open the queue.

max_msgs

The maximum number of messages if creating the queue.

mode

The open mode for the queue.

Returns

A queue ID on success, or -1 on failure and errno is set.

Description

This function opens a queue for reading or writing, as well as possibly creating it. The *max_msgs* is ignored unless the queue is being created. The possible values for *mode* are the same as the open modes for the open system call. Typically a queue should be opened for reading or writing, but not both.

See Also

Cascade QueueServer Functions, IP_QueueClose

IP_QueueRead

`IP_QueueRead` — reads a message from the queue.

Syntax

```
#include <cogent.h>
int IP_QueueRead (
   int qid,
   void* msg,
   int maxlength
);
```

Arguments

`qid`

A queue ID.

`msg`

A buffer to hold the new message.

`maxlength`

The maximum length in bytes of *`msg`*.

Returns

The number of bytes read on success, or `-1` on failure and `errno` is set.

Description

This function reads the next available message from the queue specified by *`qid`*. If no message is available, this function returns immediately. If there is insufficient space to hold the entire message then the result is operating-system dependent. (This will change in future releases).

See Also

Cascade QueueServer Functions, `IP_QueueWrite`

IP_QueueStrerror

IP_QueueStrerror — gives access to error strings.

Syntax

```
#include <cogent.h>
char* IP_QueueStrerror (
   int status
   );
```

Arguments

status

A value of errno.

Returns

A pointer to a NUL-terminated character string.

Description

This function lets you access errno strings. It returns a pointer to a string that corresponds to the errno value, as specified by *status*. This pointer must not be freed or modified by the programmer.

See Also

Cascade QueueServer Functions

IP_QueueWait

IP_QueueWait — requests notification of an event.

Syntax

```
#include <cogent.h>
int IP_QueueWait (
   int qid,
   int pulse,
   int eventmask
);
```

Arguments

qid

The queue ID.

pulse

A pulse ID.

eventmask

The event on which to wait.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function causes **qserve** to trigger the *pulse* when a particular event occurs. The calling process does not block waiting for the next event, but will return immediately. In order to implement a blocking wait, the programmer should use IP_QueueWait followed by IP_Receive.

The possible values for *eventmask* are:

- IP_NOTIFY_MSG notifies the caller when the next message is available. This event causes at most one pulse trigger.
- IP_NOTIFY_EVERY_MSG notifies the caller when every message is available. This event will cause on pulse trigger for each message that arrives in the queue.
- IP_NOTIFY_SPACE notifies the caller when the queue transitions from full to non-full. This event causes at most one pulse trigger.
- IP_NOTIFY_ANY_SPACE notifies the caller whenever the queue transitions from full to non-full.
- IP_NOTIFY_HALF_SPACE notifies the caller when the queue transitions from half-full to less than half-full. This event causes at most one pulse trigger.
- IP_NOTIFY_ANY_HALF_SPACE notifies the caller whenever the queue transitions from half-full to less than half-full.

If the condition for an event is already true at the time of the call to IP_QueueWait, then the pulse is triggered immediately.

See Also

Cascade QueueServer Functions, Pulses and Timers

IP_QueueWrite

IP_QueueWrite — writes a message to a queue.

Syntax

```
#include <cogent.h>
int IP_QueueWrite (
   int qid,
   void* msg,
   int length,
   int priority
);
```

Arguments

qid

A queue ID.

msg

The message buffer to be sent to the queue.

length

The length in bytes of *msg*.

priority

The priority of this message.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function writes a message to the queue specified by *qid*. Higher *priority* numbers are treated as having a higher priority. Priority ordering is not currently implemented. A single write to the queue is limited by the operating system. Queue messages transmitted in a single write will never be broken apart or concatenated with other messages.

See Also

Cascade QueueServer Functions, IP_QueueRead

IP_Receive

`IP_Receive` — receives any message.

Syntax

```
#include <cogent.h>
int IP_Receive (
   IP_Task* myself,
   IP_Msg* rmsg,
   IP_MsgInfo* msginfo
);
```

Arguments

`myself`

A task structure referring to the current process.

`rmsg`

A message structure to hold the received message.

`msginfo`

A structure to be filled with extra information.

Returns

The type of the received message.

Description

This function receives message of any form, and classifies them according to type. Relevant information about the sender is stored in the `msginfo` structure. A message can be one of the following types:

- `IP_GUI` means a GUI event occurred. No further processing is necessary.
- `IP_ASYNC` means an asynchronous message was received. No reply is necessary.
  - `msginfo->subtype =` `ST_DH_EXCEPTION`, `ST_DH_ECHO`, or other.
  - `msginfo->rcvid =` the queue pulse ID
- `IP_SYNC` means a synchronous message was received. A reply is necessary via a call to `IP_MsgInfoReply`.
  - `msginfo->rcvid =` the `rcvid` (QNX 6) or the `pid` (QNX 4/Linux)
- `IP_NONE` means `IP_Receive` returned without receiving a message. This is possible if the queue pulse is triggered, but the queued message is no longer available.
- `IP_ERROR` means an error occurred during `IP_Receive`, but defied classification.
  - `msginfo->rcvid = errno`
- `IP_SIGNAL` means `IP_Receive` exited due to a signal.

- `IP_PULSE` means a pulse was received. No reply is necessary.
  - `msginfo->rcvid =` the pulse ID.

- `IP_RAW` means a message from a task not using the Cascade IPC library was received. The message data is contained in `(void*)(rmsg->msg)`. A reply is required with a call to `IP_MsgInfoReplyRaw`.
  - `msginfo->rcvid =` the `rcvid` (QNX 6) or the `pid` (QNX 4/Linux)

- `IP_FD` means activity occurred on a file descriptor. No data has been read from the file descriptor.
  - `msginfo->rcvid =` the file descriptor.

See Also

Receiving Messages and Events, `IP_ReceiveNonblock`

IP_ReceiveNonblock

`IP_ReceiveNonblock` — receives any message, without blocking.

Syntax

```
#include <cogent.h>
int IP_ReceiveNonblock (
   IP_Task* myself,
   IP_Msg* rmsg,
   IP_MsgInfo* msginfo
);
```

Arguments

`myself`

A task structure referring to the current process.

`rmsg`

A message structure to hold the received message.

`msginfo`

A structure to be filled with extra information.

Returns

The type of the received message. If no message was available then `-1` is returned and `msginfo->rcvid` is set to `EAGAIN`.

Description

This function receives a message of any type, but will not block if no message is available. See `IP_Receive` for details.

See Also

Receiving Messages and Events, `IP_Receive`

IP_RemoveFDHandler

IP_RemoveFDHandler — prevents IP_Receive from accepting file input.

Syntax

```
#include <cogent.h>
int IP_RemoveFDHandler (
   int fd
   );
```

Arguments

fd

A file descriptor.

Returns

0

Description

This function removes the given file descriptor from consideration by IP_Receive.

See Also

Receiving Messages and Events, IP_AddFDHandler

IP_Reply

IP_Reply — replies to an IP_SYNC message using rcvid.

Syntax

```
#include <cogent.h>
int IP_Reply (
   int rcvid,
   IP_Msg* rmsg
);
```

Arguments

rcvid

The rcvid returned from IP_Receive in msginfo->rcvid.

rmsg

The message to transmit as a reply.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function replies to a previously recevied IP_SYNC message by using the rcvid directly. This is sometimes simpler than using the IP_MsgInfo structure, as the rcvid is a single integer that can be easily passed to other functions.

See Also

Replying to Messages, IP_MsgInfoReply

IP_ReplyRaw

IP_ReplyRaw — replies to an IP_RAW message using rcvid.

Syntax

```
#include <cogent.h>
int IP_ReplyRaw (
   int rcvid,
   char* msg,
   int len
);
```

Arguments

rcvid

The rcvid returned from IP_Receive in msginfo->rcvid.

msg

The message buffer to be transmitted.

len

The length of the message buffer.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function replies to a previously received IP_RAW message by using the rcvid directly. This is sometimes simpler than using the IP_MsgInfo structure, as the rcvid is a single integer that can be easily passed to other functions. *len* bytes of *msg* are transmitted as the reply message.

See Also

Replying to Messages, IP_MsgInfoReplyRaw

IP_SelectFD

`IP_SelectFD` — is used internally only.

Syntax

```
#include <cogent.h>
int IP_SelectFD (
   void
   );
```

Arguments

None.

Returns

Undocumented.

Description

This is an internal function. Do not call it directly.

See Also

Receiving Messages and Events

IP_SetChannelID

IP_SetChannelID — sets the channel ID for IP library use.

Syntax

```
#include <cogent.h>
void IP_SetChannelID (
   int chid
   );
```

Arguments

chid

A channel ID.

Returns

None.

Description

This function sets the channel ID to be used by the IP library. This is stored in a static variable within the library and is accessible with IP_GetChannelID. This function is normally only needed when interfacing to a facility that implements its own event loop and has already created a channel ID. Photon is such a facility.

See Also

Connections and Channels, IP_GetChannelID

IP_SetConnectionID

IP_SetConnectionID — sets the connection ID for IP library use.

Syntax

```
#include <cogent.h>
void IP_SetConnectionID (
   int coid
   );
```

Arguments

coid

A connection ID.

Returns

Nothing.

Description

This function sets the connection ID to be used by the IP library. This is stored in a static variable within the library and is accessible with IP_GetConnectionID. This function is normally only needed when interfacing to a facility that implements its own event loop and has already created a connection ID. Photon is such a facility.

See Also

Connections and Channels, IP_GetConnectionID

IP_SetGUIHandler

IP_SetGUIHandler — sets callback functions for GUI events.

Syntax

```
#include <cogent.h>
int IP_SetGUIHandler (
   IP_pfGUIFilter filter,
   IP_pfGUIHandler handler
);
```

Arguments

filter

A GUI filter function.

handler

A GUI event handler function.

Returns

0

Description

This function sets the two callback functions required to deal with a GUI event through IP_Receive. These functions must match the following types:

```
typedef int (*IP_pfGUIFilter)(int rcvid, void* msg, int length);
typedef int (*IP_pfGUIHandler)(void* msg, int length);
```

The filter function returns 0 if the *rcvid*, *msg* and *length* do not represent a GUI message for this GUI, and returns non-zero if the given message is a GUI message.

The handler function must be able to process any message for which the filter function returns non-zero, routing the message to the appropriate GUI facility.

If the filter function returns non-zero, then the IP library functions will perform no further processing on that message, but will return from IP_Receive with a message type of IP_GUI. The programmer is not required to provide code for a message of type IP_GUI.

See Also

Receiving Messages and Events, IP_Receive

IP_TaskCloseAsync

IP_TaskCloseAsync — closes queues and cleans up resources.

Syntax

```
#include <cogent.h>
int IP_TaskCloseAsync (
   IP_Task* task
   );
```

Arguments

task

A task pointer.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function closes any open queues, either for reading or writing, associated with the *task*, and cleans up any associated resources. This may result in destroying the pulse used by the queue notification services.

See Also

Task Structures, IP_TaskInitAsync, IP_TaskSendAsync, IP_TaskWaitAsync

IP_TaskCloseSync

IP_TaskCloseSync — closes synchronous connections and cleans up resources.

Syntax

```
#include <cogent.h>
int IP_TaskCloseSync (
   IP_Task* task
   );
```

Arguments

task

A task pointer.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function closes any open synchronous connections to the given task, and cleans up any associated resources. The *task* should not refer to the current process.

See Also

Task Structures, IP_TaskSendSync

IP_TaskConnect

IP_TaskConnect — opens a connection to a receiver.

Syntax

```
#include <cogent.h>
int IP_TaskConnect (
   IP_Task* receiver
   );
```

Arguments

receiver

A task to which a connection should be made.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function opens a connection between the current task and the receiver. This connection remains open until the receiver task structure is destroyed. It is not normally necessary to call this function, as the IP_TaskSend* functions do this on your behalf.

See Also

Sending Messages, IP_TaskSendSync, IP_TaskSendAsync, IP_TaskSendRaw

IP_TaskCopy

IP_TaskCopy — copies a task structure.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskCopy (
   IP_Task* task
   );
```

Arguments

task

A pointer to a task structure to be copied.

Returns

A pointer to a new task structure.

Description

This function makes a copy of the task structure pointed to by *task*. This will not perform queue initialization or name registration with **nserve**.

See Also

Task Structure Caching

IP_TaskCreate

IP_TaskCreate — creates a new task structure.

Syntax

```
#include <cogent.h>
IP_Task*IP_TaskCreate (
   int nid,
   int pid,
   int chid,
   char* name,
   char* domain,
   char* qname,
   int security,
   void* userdata
);
```

Arguments

nid

A node ID.

pid

A process ID.

chid

A channel ID.

name

The name of the task.

domain

The domain name, or NULL.

qname

A queue name, or NULL.

security

Unused, should be 0.

userdata

A pointer to user-defined data, or NULL.

Returns

The new task structure, or NULL if there was insufficient memory to perform the allocation.

Description

This function creates a new task structure, and calls IP_TaskSetInfo.

See Also

Task Structures, IP_TaskSetInfo, IP_TaskCreateMe, IP_TaskDefaultDomain, IP_TaskDestroy

IP_TaskCreateMe

IP_TaskCreateMe — creates a task structure for the current process.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskCreateMe (
   int chid,
   char* name,
   char* domain,
   char* qname,
   int qsize
);
```

Arguments

chid

A channel ID, or 0.

name

A name for this task.

domain

A Cascade DataHub domain name.

qname

A queue name, or NULL.

qsize

The maximum number of messages in the queue, or 0.

Returns

A pointer to a new task structure or NULL on failure. This function might fail if a *qname* is specified but the queue name either already exists, or could not be created.

Description

This function creates a task structure referring to the current process. If *domain* is NULL, then the default domain is used. If *qname* is non-NULL, then the queue is created and opened as read-only. If *qsize* is 0, then the default of 100 is used.

See Also

Task Structures, IP_TaskSetInfo, IP_TaskCreateMe, IP_TaskDefaultDomain, IP_TaskDestroy

IP_TaskDefaultDomain

IP_TaskDefaultDomain — returns the Cascade DataHub default domain.

Syntax

```
#include <cogent.h>
char* IP_TaskDefaultDomain (
   void
   );
```

Arguments

None.

Returns

A NUL-terminated domain name.

Description

This function returns a static NUL-terminated character string with the name of the default Cascade DataHub domain.

See Also

Task Structures, Domains in the Cascade NameServer chapter, IP_TaskSetDomain

IP_TaskDestroy

IP_TaskDestroy — closes connections and queues, removes the task and frees memory.

Syntax

```
#include <cogent.h>
void IP_TaskDestroy (
   IP_Task* task
   );
```

Arguments

task

A task pointer.

Returns

None.

Description

This function closes any open connections on the *task*, closes any open queues, removes the *task* from the task cache, and frees any memory associated with the *task* structure, including the structure itself.

See Also

Task Structures, IP_TaskCreate

IP_TaskFindID

IP_TaskFindID — finds a task by its node, process, and channel IDs.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskFindID (
   int nid,
   int pid,
   int chid
);
```

Arguments

nid

The node ID of the task to be found.

pid

The process ID of the task to be found.

chid

The channel ID of the task to be found.

Returns

A pointer to a cached IP_Task whose nid, pid and chid exactly match the requested values, or NULL if there is no match.

Description

This function looks up a task in the current process's task cache by matching the node, process ID and channel ID of the target task. Channel ID is only relevant in QNX 6.

See Also

Task Structure Caching, IP_TaskFindName

IP_TaskFindName

IP_TaskFindName — finds a task by its name.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskFindName (
   char* name
   );
```

Arguments

name

The task name to be found.

Returns

A pointer to a cached IP_Task structure if the name matches, or NULL.

Description

This function searches the current process's task cache for the specified *name*. It will return the first exact match that it encounters. Names should be unique throughout the system, though it is possible that more than one task could register a name under certain network conditions, and therefore more than one task could be cached with the same name.

See Also

Task Structure Caching, IP_TaskFindID

IP_TaskInitAsync

`IP_TaskInitAsync` — creates a read-only queue.

Syntax

```
#include <cogent.h>
int IP_TaskInitAsync (
   IP_Task* myself,
   char* qname,
   int qsize
);
```

Arguments

`myself`

A task structure referring to the current task.

`qname`

A queue name.

`qsize`

The maximum number of messages in the queue, or `0`.

Returns

`0` on success, or `-1` on failure and `errno` is set.

Description

This function creates a queue of *`qname`* and *`qsize`*, and opens it as read-only. If *`qsize`* is `0`, the default of `100` is used. This function also requests notification on the queue for every message in the queue.

See Also

Task Structures, `IP_TaskInitAsyncWrites`, `IP_TaskSendAsync`, `IP_TaskWaitAsync`, `IP_TaskCloseAsync`

IP_TaskInitAsyncWrites

`IP_TaskInitAsyncWrites` — opens a task's queue as write-only.

Syntax

```
#include <cogent.h>
int IP_TaskInitAsyncWrites (
   IP_Task* receiver
   );
```

Arguments

`receiver`

A task.

Returns

`0` on success, or `-1` on failure and `errno` is set.

Description

This function opens the *`receiver`*'s queue as write-only, allowing subsequent calls to `IP_TaskSendAsync` to succeed.

See Also

Task Structures, `IP_TaskInitAsync`

IP_TaskIntern

IP_TaskIntern — adds a task to a process's task cache.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskIntern (
   IP_Task* task
   );
```

Arguments

task

A pointer to a task structure.

Returns

Returns *task* on success, or NULL if there is already a matching task in the cache.

Description

This function causes the *task* to be added to the task cache for this process. If a task matching the nid, pid and chid of the *task* already exists in the cache, then the new task is not added to the cache, and NULL is returned.

See Also

Task Structure Caching, IP_TaskUnintern

IP_TaskNew

IP_TaskNew — creates a new task structure.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskNew (
   void
   );
```

Arguments

None.

Returns

A new task structure.

Description

This function returns a new initialized task structure.

See Also

Task Structures, IP_TaskZero

IP_TaskSendAsync

IP_TaskSendAsync — transmits a message via **qserve** and returns immediately.

Syntax

```
#include <cogent.h>
int IP_TaskSendAsync (
   IP_Task* myself,
   IP_Task* receiver,
   IP_Msg* smsg,
   int priority
);
```

Arguments

myself

A task structure referring to the current task.

receiver

The task receiving the message.

smgs

The message structure containing the data to send.

priority

The queue priority for this message..

Returns

0 on success, or -1 on failure and errno is set.

Description

This function transmits a message from *myself* to the *receiver* via **qserve**, and returns immediately. This is nominally a non-blocking call, though it is in fact implemented as a synchronous send to **qserve**, which itself is non-blocking. There is no reply from the *receiver*.

See Also

Sending Messages, IP_TaskInitAsync, IP_TaskWaitAsync, IP_TaskCloseAsync, IP_TaskSendSync, IP_TaskSendRaw

IP_TaskSendRaw

IP_TaskSendRaw — sends data in bytes, synchronously.

Syntax

```
#include <cogent.h>
int IP_TaskSendRaw (
   IP_Task* receiver,
   char* outbuf,
   int lout,
   char* inbuf,
   int lin
);
```

Arguments

myself

A task structure referring to the current task.

receiver

The task receiving the message.

outbuf

The message data to send.

lout

The number of bytes of data to send.

inbuf

The buffer to receive the reply.

lin

The maximum number of bytes in the reply.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function transmits a number of bytes synchronously to the *receiver*, and waits for the *receiver* to reply. There is no limit to how long the *receiver* may delay prior to replying. The reply data will be written into *inbuf*, up to *lin* bytes. If the *inbuf* data area is not large enough to accommodate the entire reply, then the result is OS-dependent. This will change in future versions of this API.

See Also

Sending Messages, IP_TaskSendSync, IP_TaskSendAsync

IP_TaskSendSync

IP_TaskSendSync — transmits a message, and waits for a reply.

Syntax

```
#include <cogent.h>
int IP_TaskSendSync (
   IP_Task* myself,
   IP_Task* receiver,
   IP_Msg* smsg,
   IP_Msg* rmsg
);
```

Arguments

myself

A task structure referring to the current task.

receiver

The task receiving the message.

smsg

The message structure containing the data to send.

rmsg

The message structure into which received data will be written.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function transmits a message (*smsg*) from *myself* to the *receiver*, and waits for the *receiver* to reply. There is no limit to how long the *receiver* may delay prior to replying. The reply data will be written into the *rmsg* buffer. If the *rmsg* data area is not large enough to accommodate the entire reply, then the result is OS-dependent. This will change in future versions of this API.

See Also

Sending Messages, IP_TaskCloseSync, IP_TaskSendAsync, , IP_TaskSendRaw

IP_TaskSetDomain

IP_TaskSetDomain — sets or changes a task's domain name.

Syntax

```
#include <cogent.h>
void IP_TaskSetDomain (
   IP_Task* task,
   char* domain
);
```

Arguments

task

A task.

domain

A domain name, or NULL.

Returns

None.

Description

This function sets or changes the domain name for the *task*. If the *task* has an existing domain name, then memory associated with that domain name is freed and a new domain name is allocated. If *domain* is NULL, then the default domain name is used.

See Also

Task Structures, Domains in the Cascade NameServer chapter, IP_TaskDefaultDomain, IP_TaskSetQname

IP_TaskSetInfo

IP_TaskSetInfo — sets the fields in a task structure.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskSetInfo (
   IP_Task* task,
   int nid,
   int pid,
   int chid,
   char* name,
   char* domain,
   char* qname,
   int security,
   void* userdata
);
```

Arguments

task

A task pointer.

nid

A node ID.

pid

A process ID.

chid

A channel ID.

name

The name of the task.

domain

The domain name, or NULL.

qname

A queue name, or NULL.

security

Unused, should be 0.

userdata

A pointer to user-defined data, or NULL.

Returns

task

Description

This function sets the various fields in the task structure. It is simply a convenience function. If *domain* is NULL, the default domain is used.

See Also

Task Structures, Domains in the Cascade NameServer chapter, `IP_TaskSetDomain`, `IP_TaskDefaultDomain`

IP_TaskSetQname

IP_TaskSetQname — sets a task's queue name.

Syntax

```
#include <cogent.h>
void IP_TaskSetQname (
   IP_Task* task,
   char* qname
);
```

Arguments

task

A task.

qname

A queue name, or NULL.

Returns

None.

Description

This function sets the queue name for the *task*. If the *task* already has an open queue, then that queue is closed. If *qname* is non-NULL and the task had a previously open queue, then the new queue name is opened and initialized with the same permissions.

See Also

Task Structures, Domains in the Cascade NameServer chapter, IP_TaskSetDomain

IP_TaskUnintern

IP_TaskUnintern — removes a task from a process's task cache.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskUnintern (
   IP_Task* task
   );
```

Arguments

task

A pointer to a task to be removed from cache.

Returns

Returns *task* on success, or NULL if *task* is not in the cache.

Description

This function removes the *task* from the task cache. The match is done as a pointer comparison, so the *task* must be the exact task that is currently in the cache.

See Also

Task Structure Caching, IP_TaskIntern

IP_TaskWaitAsync

IP_TaskWaitAsync — registers the task for events in **qserve**.

Syntax

```
#include <cogent.h>
int IP_TaskWaitAsync (
   IP_Task* task,
   int flags
);
```

Arguments

task

A pointer to a task structure.

flags

The event on which to wait.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function causes the current task to be notified whenever the events in flags occur within **qserve**. This function will create a pulse, stored in task->qpulse, if necessary. When the event occurs, task->qpulse will be triggered, and will subsequently be received through IP_Receive as an IP_PULSE event type.

The possible values for *flags* are:

- IP_NOTIFY_MSG notifies the caller when the next message is available. This event causes at most one pulse trigger.
- IP_NOTIFY_EVERY_MSG notifies the caller when every message is available. This event will cause on pulse trigger for each message that arrives in the queue.
- IP_NOTIFY_SPACE notifies the caller when the queue transitions from full to non-full This event causes at most one pulse trigger.
- IP_NOTIFY_ANY_SPACE notifies the caller whenever the queue transitions from full to non-full.
- IP_NOTIFY_HALF_SPACE notifies the caller when the queue transitions from half-full to less than half-full. This event causes at most one pulse trigger.
- IP_NOTIFY_ANY_HALF_SPACE notifies the caller whenever the queue transitions from half-full to less than half-full.

The programmer should not make this call with a task structure referring to the current process, as it is done automatically by the API.

See Also

Task Structures, IP_TaskInitAsync, IP_TaskSendAsync, IP_TaskCloseAsync

IP_TaskZero

IP_TaskZero — sets all task structure fields to defaults.

Syntax

```
#include <cogent.h>
IP_Task* IP_TaskZero (
   IP_Task* task
   );
```

Arguments

task

An uninitialized task structure.

Returns

task

Description

This function sets all fields in a task structure to default values. The task structure is assumed to be uninitialized, so this function does not attempt to free memory associated with strings within the task structure.

See Also

Task Structures, IP_TaskNew

IP_TimerTime

IP_TimerTime — adjusts timer parameters.

Syntax

```
#include <cogent.h>
int IP_TimerTime (
   timer_t timer,
   int absolute,
   time_t init_sec,
   time_t init_nsec,
   time_t interval_sec,
   time_t interval_nsec
);
```

Arguments

timer

A timer ID, as returned by IP_PulseTimed.

absolute

1 for absolute time, 0 for relative.

init_sec

Seconds of delay prior to the first timer expiry.

init_nsec

Nanoseconds of delay prior to the first timer expiry.

interval_sec

Seconds of delay for subsequent timing intervals.

interval_nsec

Nanoseconds of delay for subsequent timing intervals.

Returns

0 on success, or -1 on failure and errno is set.

Description

This function changes the times associated with the *timer*. If *absolute* is non-zero, then the *init_sec* and *init_nsec* fields are treated as the number of seconds and nanoseconds since 12:00 a.m. Jan 1, 1970 UTC. Otherwise, *init_sec* and *init_nsec* are treated as an offset from the current time.

See Also

Pulses and Timers, IP_PulseTimed

IP_UnselectFD

`IP_UnselectFD` — is used internally only.

Syntax

```
#include <cogent.h>
int IP_UnselectFD (
   void
   );
```

Arguments

None.

Returns

Undocumented.

Description

This is an internal function. Do not call it directly.

See Also

Receiving Messages and Events

VII. Cascade TextLogger Functions

Table of Contents

LG_Cache 250
LG_Collect 252
LG_Disable 254
LG_Empty 255
LG_Enable 256
LG_Exit 257
LG_Fall 258
LG_File 260
LG_Flush 262
LG_Group 263
LG_Log 264
LG_Output 265
LG_Time 266
LG_Timestamp 268
LG_Tolerance 270
LG_UseGMT 271

LG_Cache

LG_Cache — controls how frequently data is written.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Cache (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   int cached,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

cached

Any non-0 value sets caching on (the default). A value of 0 turns off caching.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **cache** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you determine how frequently data is written. When caching is on (*cached* is set to 0, the default), data is written in blocks, according to your default C library file buffer implementation. When caching is off (any non-zero value), each line of output is immediately flushed from the buffer to the file.

If the Cascade TextLogger is started with the -F option, this function is ignored because there is no possibility of caching; all data is immediately flushed to the file.

This function corresponds to the Cascade TextLogger **cache** command.

LG_Collect

LG_Collect — specifies when a line of data is considered complete.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Collect (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* style,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

style

One of the following:

any

Writes a line when data from any point is eligible to be written. Leaves values for all other points empty.

fill

Writes a line when data from any point is eligible to be written. Writes previous values for all other points.

all

Writes a line only after data from all points is eligible to be written.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **collect** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you specify when a line of data in a given log or group is considered complete, and ready for writing. If no *labels* are specified, this command sets a global default value for all logs and groups. Any specific value always overrides the global definition, regardless of the order in the configuration file or when a command is sent.

The Cascade TextLogger always keeps one line of output for each log in a buffer. Whenever a point changes value, the TextLogger checks each log to see if it includes that point. If so, the TextLogger checks the *style* specified for the log. If it is any or fill *and* if the tolerance for that log has been exceeded, the line in the buffer gets written immediately, and this point is entered in a new line in the buffer. But if the *style* is set to all, the TextLogger keeps the line in the buffer until every point in the log has changed at least once, *or* until a point is logged beyond the tolerance. When one of these conditions is met, the line gets written.

This function corresponds to the Cascade TextLogger **collect** command.

LG_Disable

LG_Disable — renders a log or group inactive.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Disable (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **disable** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you inactivate any number of logs or groups. This function corresponds to the Cascade TextLogger **disable** command.

See Also

LG_Enable

LG_Empty

LG_Empty — specifies a place-holder string for empty data readings.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Empty (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* emptystring
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

emptystring

The string which will replace a non-existent value when the LG_Collect style is set to fill.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **empty** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you specify a string to be written whenever there is no data available for a point. This is a global value, applied to all logs.

This function corresponds to the Cascade TextLogger **empty** command.

LG_Enable

LG_Enable — activates a log or group.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Enable (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **enable** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you reactivate a disabled log or group. A log or group is automatically enabled when the LG_Log or LG_Group function is first called, or when a **log** or **group** command is first given. This function corresponds to the Cascade TextLogger **enable** command.

See Also

LG_Disable

LG_Exit

LG_Exit — exits the Cascade TextLogger.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Exit (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   int exitstatus
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

exitstatus

The exit status to be returned to the operating system.

Returns

If this function returns a value at all, it will be NULL, indicating that the exit function failed. Normally this function will cause the Cascade TextLogger to exit, so that the client will see a disconnection rather than a return value.

Description

This function corresponds to the Cascade TextLogger **exit** command.

LG_Fall

LG_Fall — associates values logged within a tolerance.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Fall (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* direction,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

direction

One of the following:

forward

Associates a new value with the next logged value.

backward

Associates a new value with the previously logged value.

closest

Associates a new value with the value nearest in time.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **fall** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function determines how the Cascade TextLogger will associate different values logged within a given tolerance. It corresponds to the Cascade TextLogger **fall** command. Please refer to that document for an example.

LG_File

LG_File — specifies the file to receive a log.

Syntax

```
#include <cogent.h>
ST_STATUS LG_File (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* filename,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

filename

One of the following:

- The name of the file that will be logged to.
- stdout
- stderr

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **file** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you specify the filename that a log or group will be written to. If no *labels* are specified, this command sets a global default value for all logs and groups. Any specific value always overrides the global definition, regardless of the order in the configuration file or when a command is sent.

If this command specifies multiple *labels*, the logs for all of them will be written to one file. They will not necessarily always be written in exact time-sequential order, however. You may have to post-process the file if you need that kind of record.

The Cascade TextLogger writes each log only once. If you apply the `LG_File` function twice for the same log, writing to a different file each time, for example, only the most recent application of `LG_File` will produce results. Thus, if you need the same information written in two places, make two identical logs with different labels.

This function corresponds to the Cascade TextLogger **file** command.

LG_Flush

LG_Flush — flushes all buffered output.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Flush (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **flush** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function attempts to log any collected data to the cache. If data is not being cached (as determined by the LG_Cache function or the **cache** command), the LG_Flush function will cause data to be flushed to file or standard output.

This function corresponds to the Cascade TextLogger **flush** command.

LG_Group

LG_Group — groups a number of logs or groups together.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Group (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* label,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

label

The name of the group to be created.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **group** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you group any number of logs or groups. Grouping allows you to give commands that refer to several logs at the same time, because a command specifying a group will be applied to all the logs in that group or its sub-groups.

Groups can be made recursively (ie. groups of groups). Each group must contain either groups or logs, but not both; you are not permitted to have a group consisting of groups and single logs.

This function corresponds to the Cascade TextLogger **group** command.

LG_Log

LG_Log — writes point data in formatted lines.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Log (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* label,
   char* spec,
   int npoints,
   char** pointnames
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

label

The name of the log.

spec

The format string for the log line, similar to the kind used in the printf statement. Please refer to the Cascade TextLogger **log** command documentation for details.

npoints

The number of entries in the array *pointnames*.

pointnames

The names of the points corresponding to format directives in the *spec* format string.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **log** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function specifies which points to log, and the format of each line of the written output. It corresponds to the Cascade TextLogger **log** command.

LG_Output

LG_Output — writes an output string to a log or group, stdout, or stderr.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Output (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* label,
   char* output
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

label

The log or group to which the *output* will be written. You can also specify stdout to write to standard output, or stderr to write to standard error.

output

The text to write.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **output** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function inserts an output string as a line into a log or group, standard output, or standard error. If the *label* is for a single log, the string will appear in that log only. If the *label* is for a group, then the string will appear in each of the logs pertaining to that group.

This function corresponds to the Cascade TextLogger **output** command.

LG_Time

LG_Time — specifies the time format for a log or group.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Time (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* timespec,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

timespec

A time specification format string, whose options are as follows:

%J	The day of the week (e.g., Mon).
%j	The day of the month [1-31].
%M	The month (e.g., Jan).
%m	The month in the year [1-12].
%y	The year in the century (e.g., 96).
%Y	The year with century (e.g., 1996).
%h	The hour in the day [0-23].
%n	The minute in the hour [0-59].
%z	The second in the minute [0-59].
%Z	The seconds since Jan 1, 1970.
%T	The tenths of seconds in second (%z).
%H	The hundredths of seconds in second (%z).
%L	The milliseconds in second (%z).
%U	The microseconds in second (%z).
%N	The nanoseconds in second (%z).
%A	Sets a global default time spec. This default can be changed at any time, including run-time.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **time** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you specify the time format for any log or group. If no *labels* are specified, this command sets a global default value for all logs and groups. Any specific value always overrides the global definition, regardless of the order in the configuration file or when a command is sent.

You can prepend any printf-style format modifiers to the time spec field names, as appropriate for the field type. Some of these format modifiers will be ignored (e.g., %y and %Y ignore format modifiers). The string-type fields, %J and %M, respond to %s modifiers.

Sub-second fields (%T, %L, %U, %N) usually need to be zero-padded to produce readable results, as in %z.%09N.

This function corresponds to the Cascade TextLogger **time** command.

LG_Timestamp

LG_Timestamp — sets the time resolution for each line.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Timestamp (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   char* timestamp,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

timestamp

Specifies how the timestamp is to be assigned for each line. One of the following:

first

Sets the timestamp to the time the first point was logged.

last

Sets the timestamp to the time the last point was logged. This is the recommended choice.

average

Sets the timestamp to the average log time of all the points.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **timestamp** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you set the time resolution for each line. If no *labels* are specified, this command sets a global default value for all logs and groups. Any specific value always overrides the global definition, regardless of the order in the configuration file or when a command is sent.

We recommend setting the timestamp to `last`, so the lines of output will be written in a non-decreasing order. Of course, this will only be true if the data itself is ordered incrementally by times. Otherwise, you may have to post-process the file and sort it to create a non-decreasing time sequence.

This function corresponds to the Cascade TextLogger **timestamp** command.

LG_Tolerance

LG_Tolerance — sets the maximum gap between timestamps of any two points.

Syntax

```
#include <cogent.h>
ST_STATUS LG_Tolerance (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   long seconds,
   long nanoseconds
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

seconds

The number of seconds in the collection tolerance.

nanoseconds

The number of nanoseconds in the collection tolerance.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **tolerance** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function sets timestamp tolerance. The tolerance is the maximum gap between the timestamps of any two points on a line. Whenever a point is to be logged, the Cascade TextLogger checks the current log line in the buffer. If the timestamp gap between this point and any other point in the buffer has exceeded the tolerance, the existing line in the buffer is written first, and then this point is logged on a new line in the buffer.

The tolerance is globally applied to all logs all the time. There is no way to prevent it from being applied, but you can effectively ignore it by setting it to a very high number. The default tolerance is one millisecond.

This function corresponds to the Cascade TextLogger **tolerance** command. Please refer to that document for an example.

LG_UseGMT

LG_UseGMT — sets time/date strings to GMT or local time.

Syntax

```
#include <cogent.h>
ST_STATUS LG_UseGMT (
   IP_Task* myself,
   IP_Task* textlog,
   char* retbuf,
   int buflen,
   int usegmt,
   int nlabels,
   char** labels
);
```

Arguments

myself

A pointer to this task's IP_Task structure normallly generated by a call to IP_NserveInit.

textlog

A pointer to the Cascade TextLogger's IP_Task structure, normally generated by a call to IP_TaskFindName.

retbuf

A pointer to a character buffer to hold the return from the call.

buflen

The length of *retbuf*, in bytes.

usegmt

1 sets the time to GMT (the default), 0 sets the time to local.

nlabels

The number of labels (log or group names) referenced in the *labels* argument.

labels

An array of char*, each of which is a pointer to a string containing a log or group name.

Returns

ST_OK on success. Otherwise ST_ERROR, and the *retbuf* will contain a more detailed error message (see **usegmt** in the Cascade TextLogger documentation). If the return value is ST_OK, the *retbuf* may not contain useful information.

Description

This function lets you set the time for the time and date strings of any log or group to Greenwich Mean Time or local time. If no *label* is specified, this command sets a global default value for all logs and groups. Any specific value always overrides the global definition, regardless of the order in the configuration file or when a command is sent.

This function corresponds to the Cascade TextLogger **usegmt** command.

VIII. Point Manipulation Functions

Table of Contents

PT_FindCPoint 273
PT_InitClient 274
PT_NewCPoint 275
PT_PointCopyValue 276
PT_PointFormat 277
PT_PointInt 278
PT_PointReal 279
PT_PointString 280

PT_FindCPoint

PT_FindCPoint — looks up a point by name in the internal point hash table.

Syntax

```
#include <cogent.h>
PT_pCPOINT PT_FindCPoint (
   char* name
   );
```

Arguments

name

The point name to be found in the internal hash table created by PT_InitClient.

Returns

A pointer to a PT_stCPOINT structure, or NULL if PT_InitClient has not yet been called.

Description

This function looks up a point by name in the internal point hash table. If the point name does not exist, then a new point structure is created through a call to PT_NewCPoint, and the result is returned.

See Also

Point Structure, Storage, and Manipulation, PT_InitClient, PT_NewCPoint

PT_InitClient

PT_InitClient — initializes the static hash table of referenced points.

Syntax

```
#include <cogent.h>
ST_STATUS PT_InitClient (
   void
   );
```

Arguments

None.

Returns

ST_OK on success, ST_ERROR on failure.

Description

This function initializes a statically held hash table that is used to maintain the definitions of all points that have been accessed by the program.

See Also

Point Structure, Storage, and Manipulation, PT_FindCPoint,

PT_NewCPoint

PT_NewCPoint — allocates and initializes a new point structure.

Syntax

```
#include <cogent.h>
PT_pCPOINT PT_NewCPoint (
   void
   );
```

Arguments

None

Returns

A newly allocated point structure with all of its members set to sane values.

Description

This function allocates a new point structure on the heap and sets all of its members to sane values.

See Also

Point Structure, Storage, and Manipulation, PT_FindCPoint

PT_PointCopyValue

PT_PointCopyValue — copies one point structure into another.

Syntax

```
#include <cogent.h>
void PT_PointCopyValue (
   PT_pCPOINT dest,
   PT_pCPOINT src
);
```

Arguments

dest

The destination of the copy.

src

The source of the copy.

Returns

Nothing.

Description

This function copies all relevant information from one point structure to another. If the destination point is of type PT_TYPE_STRING, then free() will be called on its string value prior to the copy. If the source point is of PT_TYPE_STRING, then the value of the destination will be the result of a strdup() call on the source point's string value.

PT_PointFormat

PT_PointFormat — formats a point according to a printf-like directive.

Syntax

```
#include <cogent.h>
char* PT_PointFormat (
   PT_pCPOINT point,
   char* format,
   char* buffer
);
```

Arguments

point

A point whose value should be formatted.

format

A printf-style format directive used to format the point value. Only the actual formatting directive, beginning with % and ending with one of [sfgGeEdicouxX] is acceptable. If this argument is NULL, then default formatting will be done.

buffer

A character buffer to be filled with the formatting result. It is the programmer's responsibility to ensure that the buffer is large enough to hold the formatted point value.

Returns

A pointer to the character following the formatting directive within the format string. This is useful for self-parsing a longer printf-style formatting string which contains literal strings and multiple point values.

Description

This function examines the formatting directive to determine the type of point expected, and calls one of Pt_PointString, PT_PointInt or PT_PointReal depending on the formatting directive type:

- [gGeEf] - PT_PointReal
- [diuxX] - PT_PointInt
- [s] - PT_PointString

See Also

Point Structure, Storage, and Manipulation, PT_PointString, PT_PointInt, and PT_PointReal

PT_PointInt

PT_PointInt — converts a point to an integer.

Syntax

```
#include <cogent.h>
long PT_PointInt (
   PT_pCPOINT point
   );
```

Arguments

point

A point whose value should be returned.

Returns

An integer representation of the point. value

Description

This function attempts to find the most reasonable integer representation for the given point. This will depend on the input type of the point:

- PT_TYPE_INTEGER - the point value
- PT_TYPE_REAL - the floor of the point value
- PT_TYPE_STRING - atol() of the point value

See Also

Point Structure, Storage, and Manipulation, PT_PointString, PT_PointReal, PT_PointFormat

PT_PointReal

PT_PointReal — converts a point to a real.

Syntax

```
#include <cogent.h>
double PT_PointReal (
   PT_pCPOINT point
   );
```

Arguments

point

A point whose value should be returned.

Returns

A floating point representation of the point value.

Description

This function attempts to find the most reasonable integer representation for the given point. This will depend on the input type of the point:

- PT_TYPE_INTEGER - the point value
- PT_TYPE_REAL - the point value
- PT_TYPE_STRING - atof() of the point value

See Also

Point Structure, Storage, and Manipulation, PT_PointString, PT_PointInt, and PT_PointFormat

PT_PointString

PT_PointString — converts a point to a string.

Syntax

```
#include <cogent.h>
char* PT_PointString (
   PT_pCPOINT point,
   char* format,
   char* buffer
);
```

Arguments

point

The point whose value is to be returned.

format

A printf-style format directive used to format the point value. Only the actual formatting directive, beginning with % and ending with one of [sfgGeEdicouxX] is acceptable. If this argument is NULL, then default formatting will be done.

buffer

A character buffer to be filled with the formatting result. It is the programmer's responsibility to ensure that the buffer is large enough to hold the formatted point value.

Returns

A pointer to a buffer containing the formatted value. This may or may not be a pointer to the input buffer. In any case, it is not subject to memory management (free or realloc), though buffer may be.

Description

This function attempts to produce a character string that represents the point value. Integer and real values will be written to the buffer using sprintf along with the format string. If no format string is given, then %g is used for real, and %d is used for integer. If %s is given to a point which not of type string, then the value will first be formatted into a string using the default style, and then formatted again using the %s directive in order to achieve the necessary spacing (e.g., formatting the integer 5 to "%-s" will generate a right-justified string as "5"). The return value will be either a pointer to the point value (in the case of an unformatted point of type PT_TYPE_STRING), or a pointer to the argument buffer.

See Also

Point Structure, Storage, and Manipulation, PT_PointInt, PT_PointReal, and PT_PointFormat

Index

D

DH_AppendString, 64
DH_CreatePoint, 65
DH_FindPointAddress, 66
DH_FormatPoint, 67
DH_ParsePointMsg, 69
DH_ParsePointString, 71
DH_PointAdd, 73
DH_PointDivde, 73
DH_PointMultiply, 73
DH_ReadExistingPoint, 75
DH_ReadPoint, 75
DH_RegisterAllPoints, 77
DH_RegisterPoint, 78
DH_SendPointMessage, 80
DH_SetLock, 81
DH_SetReceiveFormat, 82
DH_SetSecurity, 81
DH_SetTransmitFormat, 84
DH_UnregisterPoint, 85
DH_WriteExistingPoint, 86
DH_WriteExistingPoints, 86
DH_WritePoint, 86
DR_ApCloseIPC, 89
DR_ApCommand, 90
DR_ApConnectIPC, 91
DR_ApDescribeBuffer, 92
DR_ApDescribePnt, 94
DR_ApInitIPC, 96
DR_ApListBuffers, 97
DR_ApListPoints, 99
DR_ApPointBufAddress, 101
DR_ApReadBlock, 103
DR_ApReadControl, 105
DR_ApReadPoint, 107
DR_ApReadStatus, 109
DR_ApUpdateBuffers, 111
DR_ApWriteBlock, 112
DR_ApWriteControl, 114
DR_ApWritePoint, 116

G

gsend, 51

H

HI_Add, 118
HI_BufferIDDestroy, 120
HI_BufferIDLength, 121
HI_BufferIDRead, 122
HI_Bufsize, 124
HI_ClipBuffer, 126
HI_Count, 127
HI_Deadband, 128
HI_Delete, 131
HI_Describe, 132
HI_Disable, 134
HI_Earliest, 135
HI_Enable, 136
HI_ExchangeBuffer, 137
HI_FileBase, 138
HI_Flush, 140
HI_GapCountBuffer, 141
HI_GapFillBuffer, 142
HI_History, 143
HI_Interpolate, 145
HI_InterpolateData, 148
HI_Latest, 150
HI_Length, 151
HI_List, 152
HI_Register, 156
HI_ScaleBuffer, 154
HI_StatBuffer, 155
HI_stVALUE, 59
HI_Unregister, 157
HI_Version, 158

I

IP_AddFDHandler, 162
IP_AttachPhoton, 163
IP_AttachPhotonMainloop, 164
IP_ConnectToPort, 165
IP_ConnectToService, 166
IP_DetachPhotonMainloop, 167
IP_GetChannelID, 168
IP_GetConnectionID, 169
IP_IsPulse, 170
IP_ListenToPort, 171
IP_ListenToService, 172
IP_MsgCascade, 173
IP_MsgCreate, 174
IP_MsgData, 175
IP_MsgDefaultSize, 176
IP_MsgDestroy, 177
IP_MsgInfoReply, 178
IP_MsgInfoReplyRaw, 179

IP_MsgLisp, 180
IP_MsgRaw, 181
IP_MsgRawData, 182
IP_MsgResize, 183
IP_NserveAdd, 184
IP_NserveClose, 185
IP_NserveInit, 186
IP_NserveInitMyself, 187
IP_NserveLookup, 188
IP_NserveLookupId, 189
IP_NserveLookupName, 190
IP_NservePackTaskInfo, 191
IP_NserveQueryNameCount, 193
IP_NserveQueryNames, 194
IP_NserveReattach, 195
IP_NserveRemove, 196
IP_NserveSetDomain, 197
IP_pfTaskComp, 198
IP_PhotonGUIFilter, 199
IP_PhotonGUIHandler, 200
IP_ProcessMessage, 201
IP_PulseDestroy, 202
IP_PulseNew, 203
IP_PulseTimed, 204
IP_PulseTrigger, 205
IP_QueueClose, 206
IP_QueueOpen, 207
IP_QueueRead, 208
IP_QueueStrerror, 209
IP_QueueWait, 210
IP_QueueWrite, 212
IP_Receive, 213
IP_ReceiveNonblock, 215
IP_RemoveFDHandler, 216
IP_Reply, 217
IP_ReplyRaw, 218
IP_SelectFD, 219
IP_SetChannelID, 220
IP_SetConnectionID, 221
IP_SetGUIHandler, 222
IP_TaskCloseAsync, 223
IP_TaskCloseSync, 224
IP_TaskConnect, 225
IP_TaskCopy, 226
IP_TaskCreate, 227
IP_TaskCreateMe, 228
IP_TaskDefaultDomain, 229
IP_TaskDestroy, 230
IP_TaskFindID, 231
IP_TaskFindName, 232
IP_TaskInitAsync, 233
IP_TaskInitAsyncWrites, 234
IP_TaskIntern, 235
IP_TaskNew, 236
IP_TaskSendAsync, 237
IP_TaskSendRaw, 238
IP_TaskSendSync, 239
IP_TaskSetDomain, 240
IP_TaskSetInfo, 241
IP_TaskSetQname, 243
IP_TaskUnintern, 244
IP_TaskWaitAsync, 245
IP_TaskZero, 246
IP_TimerTime, 247
IP_UnselectFD, 248

L

LG_Cache, 250
LG_Collect, 252
LG_Disable, 254
LG_Empty, 255
LG_Enable, 256
LG_Exit, 257
LG_Fall, 258
LG_File, 260
LG_Flush, 262
LG_Group, 263
LG_Log, 264
LG_Output, 265
LG_Time, 266
LG_Timestamp, 268
LG_Tolerance, 270
LG_UseGMT, 271
lsend, 51

N

nserve, 55
nsnames, 53

P

PT_FindCPoint, 273
PT_InitClient, 274
PT_NewCPoint, 275
PT_PointCopyValue, 276
PT_PointFormat, 277
PT_PointInt, 278
PT_PointReal, 279
PT_PointString, 280
PT_stCPOINT, 60
PT_TYPE, 61

PT_uVALUE, 61

Q

qserve, 57

S

ST_STATUS, 62

Colophon

This book was produced by Cogent Real-Time Systems, Inc. from a single-source group of SGML files. Gnu Emacs was used to edit the SGML files. The DocBook DTD and related DSSSL stylesheets were used to transform the SGML source into HTML, PDF, and QNX Helpviewer output formats. This processing was accomplished with the help of OpenJade, JadeTeX, Tex, and various scripts and makefiles. Details of the process are described in our book: Preparing Cogent Documentation, which is published on-line at
`http://developers.cogentrts.com/cogent/prepdoc/book1.html`.

Text written by Andrew Thomas, Manuel Dias, and Bob McIlvride.