HP Operations Manager for UNIX Developer's Toolkit

Developer's Reference

Software Version: 9.00

Manufacturing Part Number: None

Date: June, 2009

© Copyright 1999-2009 Hewlett-Packard Development Company, L.P.

Legal Notices

Warranty.

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices.

©Copyright 2005-2009 Hewlett-Packard Development Company, L.P.

No part of this document may be copied, reproduced, or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this material is subject to change without notice.

Trademark Notices.

Adobe® is a trademark of Adobe Systems Incorporated.

Intel®, Itanium®, and Pentium® are trademarks of Intel Corporation in the U.S. and other countries.

Java™ is a US trademark of Sun Microsystems, Inc.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered trademark of Oracle Corporation and/or its affiliates.

UNIX® is a registered trademark of the Open Group.

Contents

1. Introduction

In this Chapter . . . 28
The HPOM Application Programming Interfaces . . . 29
Function-Naming Conventions . . . 30
Libraries for HPOM Integrations . . . 34
Libraries on the Management Server . . . 35
Include Files and Makefiles . . . 36
Using APIs in Internationalized Environments . . . 37
HP Software Partnerships . . . 38

2. Functions of the HPOM Operator APIs

In this Chapter . . . 40
Data API . . . 41
Usage . . . 41
Prerequisites . . . 41
Multithread Usage . . . 41
The HPOM Container . . . 42
The HPOM Iterator . . . 43
Registration Conditions of the HPOM Data API . . . 44
opcdata_append_element() . . . 45
opcdata_clear() . . . 46
opcdata_copy() . . . 47
opcdata_copy_info_to_actresp() . . . 48
opcdata_create() . . . 49
opcdata_delete_element() . . . 50
opcdata_free() . . . 51
opcdata_generate_id() . . . 52
opcdata_get_cma() . . . 53
opcdata_get_cmanames() . . . 55
opcdata_get_double() . . . 57
opcdata_get_element() . . . 58
opcdata_get_error_msg() . . . 59
opcdata_get_long() . . . 60
opcdata_get_str() . . . 61
opcdata_insert_element() . . . 62
opcdata_ladd() . . . 63
opcdata_ldel() . . . 64
opcdata_lget_len() . . . 65

Contents

opcdata_lget_long() . . . 66
opcdata_lget_str() . . . 67
opcdata_lset_long() . . . 68
opcdata_lset_str() . . . 69
opcdata_num_elements() . . . 70
opcdata_remove_cma() . . . 71
opcdata_report_error() . . . 72
opcdata_set_cma() . . . 73
opcdata_set_double() . . . 74
opcdata_set_long() . . . 75
opcdata_set_str() . . . 76
opcdata_type() . . . 77
opciter_begin() . . . 78
opciter_create() . . . 79
opciter_end() . . . 80
opciter_free() . . . 81
opciter_get_pos() . . . 82
opciter_next() . . . 83
opciter_nth() . . . 84
opciter_prev() . . . 85
opciter_set_pos() . . . 86
opcreg_copy() . . . 87
opcreg_create() . . . 88
opcreg_free() . . . 89
opcreg_get_long() . . . 90
opcreg_get_str() . . . 91
opcreg_set_long() . . . 92
opcreg_set_str() . . . 93
Interface API . . . 94
HPOM Interfaces . . . 94
Message Stream Interface . . . 94
Configuration Stream Interface . . . 96
Message Event Interface . . . 97
Application Response Interface . . . 97
Prerequisites . . . 98
Multi-thread Usage . . . 98
Registration Conditions of the HPOM Interface API . . . 98
Security Considerations for the HPOM Interface API . . . 99

opcif_close() . . . 101
opcif_close_and_buffer() . . . 103
opcif_get_pipe() . . . 104
opcif_open() . . . 106
opcif_read() . . . 112
opcif_register() . . . 114
opcif_unregister() . . . 117
opcif_write() . . . 118
opcreg_copy() . . . 120
opcreg_create() . . . 121
opcreg_free() . . . 122
opcreg_get_long() . . . 123
opcreg_get_str() . . . 124
opcreg_set_long() . . . 125
opcreg_set_str() . . . 126
Server Message API . . . 127
Data Structures . . . 127
Usage . . . 127
Prerequisites . . . 128
Multithread Usage . . . 128
opcanno_add() . . . 129
opcanno_delete() . . . 131
opcanno_get_list() . . . 133
opcanno_modify() . . . 135
opcmsg_ack() . . . 137
opcmsg_disown() . . . 139
opcmsg_get() . . . 141
opcmsg_get_instructions() . . . 143
opcmsg_get_msgids() . . . 144
opcmsg_modify() . . . 146
opcmsg_own() . . . 148
opcmsg_select() . . . 150
opcmsg_set_owner() . . . 152
opcmsg_start_auto_action() . . . 154
opcmsg_start_op_action() . . . 156
opcmsg_unack() . . . 158
opcmsg_unbuffer() . . . 160
Agent Message API . . . 162

Contents

Data Structures 162
Usage 162
Prerequisites 162
Multithread Usage 162
Agent Configuration 163
opcagtmsg_ack() 164
opcagtmsg_send() 165
opcmsg() 167
Agent Monitor API 169
Data Structures 169
Usage 169
Prerequisites 169
Multithread Usage 169
opcagtmon_send() 170
opcmon() 172

3. Functions of the HPOM Configuration APIs

In This Chapter 176
Configuration API Usage 177
Example of an Object Modification 178
Connection API 179
Data Structures 179
Usage 179
Prerequisites 180
Multithread Usage 180
opc_connect() 181
opc_disconnect() 183
opcconn_get_capability() 184
opcconn_set_capability() 187
Application Configuration API 189
Data Structures 189
Usage 189
Prerequisites 189
Multithread Usage 189
opcappl_add() 190
opcappl_delete() 192
opcappl_get() 194
opcappl_get_list() 196

opcappl_modify() 198
opcappl_start() 200
Application Group Configuration API 202
Data Structures 202
Usage 202
Prerequisites 202
Multithread Usage 202
opcapplgrp_add() 203
opcapplgrp_assign_applgrps() 205
opcapplgrp_assign_appls() 207
opcapplgrp_deassign_applgrps() 209
opcapplgrp_deassign_appls() 211
opcapplgrp_delete() 213
opcapplgrp_get() 215
opcapplgrp_get_applgrps() 217
opcapplgrp_get_appls() 219
opcapplgrp_get_list() 221
opcapplgrp_modify() 223
Category Configuration API 225
Data Structure 225
Return Values 226
Restrictions 226
Multithread Usage 226
opcinstrum_add_categories() 227
opcinstrum_del_categories() 229
opcinstrum_get_categories() 231
opcinstrum_get_category() 232
opcinstrum_modify_categories() 233
opcpolicy_assign_categories() 235
opcpolicy_deassign_categories() 237
opcpolicy_get_categories() 239
opcnode_assign_categories() 241
opcnode_deassign_categories() 243
opcnode_get_categories() 245
Instruction Text Interface Configuration API 247
Data Structures 247
Usage 247
Prerequisites 248

Contents

Multithread Usage 248
opcinstruction_add() 249
opcinstruction_del() 251
opcinstruction_get() 252
opcinstruction_modify() 254
Message Group Configuration API 256
Data Structures 256
Usage 256
Prerequisites 256
Multithread Usage 256
opcmsggrp_add() 257
opcmsggrp_delete() 259
opcmsggrp_get() 261
opcmsggrp_get_list() 262
opcmsggrp_modify() 263
Message Regroup Condition Configuration API 265
Data Structures 265
Usage 265
Prerequisites 265
Multithread Usage 265
opcmsgregrp_add() 266
opcmsgregrp_delete() 268
opcmsgregrp_get() 270
opcmsgregrp_get_list() 272
opcmsgregrp_modify() 274
opcmsgregrp_move() 276
Node Configuration API 278
Data Structures 278
Usage 279
Prerequisites 279
Multithread Usage 279
opcnode_add() 280
opcnode_assign_templates() 282
opcnode_deassign_templates() 284
opcnode_delete() 286
opcnode_get() 288
opcnode_get_defaults() 290
opcnode_get_list() 292

opcnode_get_templates() 293
opcnode_modify(). 295
opcnode_modify_defaults() 297
opcnodegrp_add() 299
opcnodegrp_assign_nodes() 301
opcnodegrp_assign_templates() 303
opcnodegrp_deassign_nodes() 305
opcnodegrp_deassign_templates() 307
opcnodegrp_delete() 309
opcnodegrp_get() 311
opcnodegrp_get_list() 313
opcnodegrp_get_nodes() 315
opcnodegrp_get_templates() 317
opcnodegrp_modify() 319
Node Hierarchy Configuration API. 321
Data Structures 321
Usage 322
Prerequisites 322
Multithread Usage 322
opcnodehier_add() 323
opcnodehier_add_layoutgrp() 325
opcnodehier_copy() 327
opcnodehier_delete() 329
opcnodehier_delete_layoutgrp() 330
opcnodehier_get() 332
opcnodehier_get_all_layoutgrps() 334
opcnodehier_get_all_nodes(). 336
opcnodehier_get_layoutgrp() 338
opcnodehier_get_layoutgrps(). 340
opcnodehier_get_list(). 342
opcnodehier_get_nodeparent() 344
opcnodehier_get_nodes(). 346
opcnodehier_modify() 348
opcnodehier_modify_layoutgrp() 350
opcnodehier_move_layoutgrp(). 352
opcnodehier_move_layoutgrps(). 354
opcnodehier_move_nodes() 356
Notification Schedule Configuration API 358

Contents

Data Structures 358
Usage 358
Prerequisites 359
Multithread Usage 359
opcnotischedule_add() 360
opcnotischedule_del() 362
opcnotischedule_get() 364
opcnotischedule_get_list() 366
opcnotischedule_modify() 367
Notification Service Configuration API 369
Data Structures 369
Usage 369
Prerequisites 370
Multithread Usage 370
opcnotiservice_add() 371
opcnotiservice_del() 373
opcnotiservice_get() 374
opcnotiservice_get_list() 376
opcnotiservice_modify() 377
Policy Configuration API 379
Data Structures 379
Usage 380
Prerequisites 380
Multithread Usage 380
Policy Type API 381
Policy Configuration API 395
Trouble Ticket Configuration API 420
Usage 420
Prerequisites 420
Multithread Usage 420
opctroubleticket_get() 421
opctroubleticket_set() 422
User Profile Configuration API 423
Data Structures 423
Usage 423
Prerequisites 423
Multithread Usage 423
opcprofile_add() 424

opcprofile_assign_applgrps() 426
opcprofile_assign_appls() 428
opcprofile_assign_profiles() 430
opcprofile_assign_resps() 432
opcprofile_deassign_applgrps() 434
opcprofile_deassign_appls() 436
opcprofile_deassign_profiles() 438
opcprofile_deassign_resps() 440
opcprofile_delete() 442
opcprofile_get() 444
opcprofile_get_applgrps() 446
opcprofile_get_appls() 448
opcprofile_get_list() 450
opcprofile_get_profiles() 452
opcprofile_get_resps() 454
opcprofile_modify() 456
User Configuration API 458
Data Structures 458
Usage 458
Prerequisites 458
Multithread Usage 458
opcuser_add() 459
opcuser_assign_applgrps() 461
opcuser_assign_appls() 463
opcuser_assign_nodehier() 465
opcuser_assign_profiles() 467
opcuser_assign_resps() 469
opcuser_deassign_applgrps() 471
opcuser_deassign_appls() 473
opcuser_deassign_profiles() 475
opcuser_deassign_resps() 477
opcuser_delete() 479
opcuser_get() 481
opcuser_get_applgrps() 483
opcuser_get_appls() 485
opcuser_get_list() 487
opcuser_get_nodehier() 489
opcuser_get_profiles() 491

Contents

opcuser_get_resps() . . . 493
opcuser_modify() . . . 495
Distribution API . . . 497
Data Structures . . . 497
Usage . . . 497
Prerequisites . . . 497
Multithread Usage . . . 497
opc_distrib() . . . 498
Server Synchronization API . . . 500
Data Structures . . . 500
Usage . . . 500
Prerequisites . . . 500
Multithread Usage . . . 500
The Transaction Concept . . . 501
Transaction Rules . . . 502
opc_inform_user() . . . 503
opc_version() . . . 504
opcsync_inform_server() . . . 505
opctransaction_commit () . . . 507
opctransaction_rollback() . . . 508
opctransaction_start () . . . 509
Pattern Matching API . . . 510
Usage . . . 510
Prerequisites . . . 510
Multithread Usage . . . 510
opcpat_match() . . . 511

4. Examples

In this Chapter . . . 514
Examples . . . 515
Examples of the HPOM Interfaces . . . 515
Example of the Server Message API . . . 532

5. HPOM Data Structures

In This Chapter . . . 540
HPOM Data Structures . . . 541
OPCDTYPE_CONTAINER . . . 542
OPCDTYPE_ACTION_REQUEST . . . 543

OPCDTYPE_ACTION_RESPONSE 545
OPCDTYPE_ANNOTATION 547
OPCDTYPE_APPL_CONFIG 548
OPCDTYPE_APPL_GROUP 551
OPCDTYPE_APPLIC 552
OPCDTYPE_APPLIC_RESPONSE 553
OPCDTYPE_CATEGORY_INFO 554
OPCDTYPE_CONFIG_EVENT 555
OPCDTYPE_INFORM_USER 558
OPCDTYPE_INSTR_IF 559
OPCDTYPE_LAYOUT_GROUP 560
OPCDTYPE_MESSAGE 561
OPCDTYPE_MESSAGE_EVENT 568
OPCDTYPE_MESSAGE_GROUP 571
OPCDTYPE_MESSAGE_ID 572
OPCDTYPE_MONITOR_MESSAGE 573
OPCDTYPE_NODE 574
OPCDTYPE_NODE_CONFIG 576
OPCDTYPE_NODE_GROUP 580
OPCDTYPE_NODEHIER 581
OPCDTYPE_NOTI_SCHEDULE 582
OPCDTYPE_NOTI_SERVICE 583
OPCDTYPE_REGROUP_COND 584
OPCDTYPE_TEMPLATE_INFO 585
OPCDTYPE_USER_CONFIG 586
OPCDTYPE_USER_RESP_ENTRY 587
OPCREGCOND 588

6. Service Navigator Interfaces and APIs

In this Chapter 590
The XML Data Interface 591
XML Notation Used 592
The Operations Tags 596
The Results Tags 607
The Loggings Tags 617
Return Values 618
Service Engine C++ APIs 620
The Classes 620

Contents

The Service Operations Interface Classes 623
The Registration Interface Classes 628
Allowing Remote Access to the Service Engine 636

A. About HPOM Man Pages

In this Appendix 638
Accessing and Printing Man Pages 639
To Access an HPOM Man Page from the Command Line 639
To Print a Man Page from the Command Line 639
To Access the Man Pages in HTML Format 639
Man Pages in HPOM 640
Man Pages for HPOM APIs 644
Man Pages for HP Operations Service Navigator 645

B. API Changes

In this Appendix 648
Changes from HPOM 8.xx to 9.00 649
API Changes between 8.xx and 9.00 649
Data Structure Changes between 8.xx and 9.00 654
APIs Used for Backward Compatibility 654

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

First Edition: June 2009

Conventions

The following typographical conventions are used in this manual.

Table 1 Typographical Conventions

Font	Meaning	Example
Italic	Book or manual titles, and man page names	Refer to the *HPOM Administrator's Reference* and the *opc(1M)* manpage for more information.
	Emphasis	You *must* follow these steps.
	Variable that you must supply when entering a command	At the prompt, enter **`rlogin`** ***`username`***.
	Parameters to a function	The *`oper_name`* parameter returns an integer response.
Bold	New terms	The **HTTPS agent** observes...
`Computer`	Text and other items on the computer screen	The following system message displays: `Are you sure you want to remove current group?`
	Command names	Use the `grep` command ...
	Function names	Use the `opc_connect()` function to connect ...
	File and directory names	`/opt/OV/bin/OpC/`
	Process names	Check to see if `opcmona` is running.
	Window/dialog-box names	In the `Add Logfile` window ...
	Menu name followed by a colon (`:`) means that you select the menu, then the item. When the item is followed by an arrow (`->`), a cascading menu follows.	Select `Actions: Filtering -> All Active Messages` from the menu bar.

Table 1 Typographical Conventions (Continued)

Font	Meaning	Example
`Computer Bold`	Text that you enter	At the prompt, enter **`ls -l`**
Keycap	Keyboard keys	Press **Return**.
`[Button]`	Buttons in the user interface	Click `[OK]`.

HPOM Documentation Map

HP Operations Manager (HPOM) provides a set of manuals and online help that help you to use the product and to understand the concepts underlying the product. This section describes what information is available and where you can find it.

Electronic Versions of the Manuals

All the manuals are available as Adobe Portable Document Format (PDF) files in the documentation directory on the HPOM product CD-ROM.

With the exception of the *HPOM Software Release Notes*, all the manuals are also available in the following HPOM web-server directory:

`http://<management_server>:3443/ITO_DOC/<lang>/manuals/*.pdf`

In this URL, <*management_server*> is the fully-qualified hostname of your management server, and <*lang*> stands for your system language, for example, `C` for the English environment.

Alternatively, you can download the manuals from the following website:

`http://support.openview.hp.com/selfsolve/manuals`

Watch this website regularly for the latest edition of the *HPOM Software Release Notes*, which gets updated every 2-3 months with the latest news such as additionally supported OS versions, latest patches and so on.

HPOM Manuals

This section provides an overview of the HPOM manuals and their contents.

Table 2 HPOM Manuals

Manual	Description	Media
HPOM Installation Guide for the Management Server	Designed for administrators who install HPOM software on the management server and perform the initial configuration. This manual describes: • Software and hardware requirements • Software installation and de-installation instructions • Configuration defaults	PDF only
HPOM Concepts Guide	Provides you with an understanding of HPOM on two levels. As an operator, you learn about the basic structure of HPOM. As an administrator, you gain an insight into the setup and configuration of HPOM in your own environment.	PDF only
HPOM Administrator's Reference	Designed for administrators who install HPOM on the managed nodes and are responsible for HPOM administration and troubleshooting. Contains conceptual and general information about the HPOM managed nodes.	PDF only
HPOM HTTPS Agent Concepts and Configuration Guide	Provides platform-specific information about each HTTPS-based managed-node platform.	PDF only
HPOM Reporting and Database Schema	Provides a detailed description of the HPOM database tables, as well as examples for generating reports from the HPOM database.	PDF only
HPOM Entity Relationship Diagrams	Provides you with an overview of the relationships between the tables and the HPOM database.	PDF only

Table 2 HPOM Manuals (Continued)

Manual	Description	Media
HPOM Java GUI Operator's Guide	Provides you with a detailed description of the HPOM Java-based operator GUI and the Service Navigator. This manual contains detailed information about general HPOM and Service Navigator concepts and tasks for HPOM operators, as well as reference and troubleshooting information.	PDF only
Service Navigator Concepts and Configuration Guide	Provides information for administrators who are responsible for installing, configuring, maintaining, and troubleshooting the HP Operations Service Navigator. This manual also contains a high-level overview of the concepts behind service management.	PDF only
HPOM Software Release Notes	Describes new features and helps you: • Compare features of the current software with features of previous versions. • Determine system and software compatibility. • Solve known problems.	PDF only
Managing Your Network with HP Network Node Manager	Designed for administrators and operators. This manual describes the basic functionality of the HP Network Node Manager, which is an embedded part of HPOM.	PDF only
HPOM Firewall Concepts and Configuration Guide	Designed for administrators. This manual describes the HPOM firewall concepts and provides instructions for configuring the secure environment.	PDF only
HPOM Web Services Integration Guide	Designed for administrators and operators. This manual describes the HPOM Web Services integration.	PDF only
HPOM Security Advisory	Designed for administrators. This manual describes the the HPOM security concepts and provides instructions for configuring the secure environment.	PDF only

Additional HPOM-Related Products

This section provides an overview of the HPOM-related manuals and their contents.

Table 3 Additional HPOM-related Manuals

Manual	Description	Media
HP Operations Manager for UNIX Developer's Toolkit If you purchase the HP Operations Manager for UNIX Developer's Toolkit, you receive the full HPOM documentation set, as well as the following manuals:		
HPOM Application Integration Guide	Suggests several ways in which external applications can be integrated into HPOM.	Hardcopy PDF
HPOM Developer's Reference	Provides an overview of all the available application programming interfaces (APIs).	Hardcopy PDF
HP Event Correlation Designer for HPOM If you purchase HP Event Correlation Designer for HPOM, you receive the following additional documentation. Note that HP Event Correlation Composer is an integral part of HPOM. HP Composer usage in the HPOM context is described in the OS-SPI documentation.		
HP ECS Configuring Circuits for HPOM	Explains how to use the ECS Designer product in the HPOM environment.	Hardcopy PDF

HPOM Online Information

The following information is available online.

Table 4 HPOM Online Information

Online Information	Description
HPOM Java GUI Online Information	HTML-based help system for the HPOM Java-based operator GUI and Service Navigator. This help system contains detailed information about general HPOM and Service Navigator concepts and tasks for HPOM operators, as well as reference and troubleshooting information.
HPOM Man Pages	Manual pages available online for HPOM. These manual pages are also available in HTML format. To access these pages, go to the following location (URL) with your web browser: `http://<management_server>:3443/ITO_MAN` In this URL, the variable <*management_server*> is the fully-qualified hostname of your management server. Note that the man pages for the HP Operations HTTPS agents are installed on each managed node.

1 Introduction

In this Chapter

This chapter provides information about:

- The HPOM Application Programming Interfaces
- Function-Naming Conventions
- Libraries for HPOM Integrations
- Using APIs in Internationalized Environments
- HP Software Partnerships

The HPOM Application Programming Interfaces

The HP Operations Manager Developer's Toolkit comes with a set of Application Programming Interfaces (APIs) that can be split into two logical groups:

- **HPOM Operator API**

  The HPOM Operator API contains functions that let you work on HPOM events, for example, messages, message events, or application responses. It lets you perform actions similar to using the HPOM Operator GUI, like getting messages, acknowledging messages, owning or disowning messages, or adding message annotations.

  The functions for using the **HPOM Interfaces** are also part of this API, because the interfaces are also used to get and modify HPOM events. See "Interface API" on page 94 for more information.

  See also Chapter 2, "Functions of the HPOM Operator APIs," on page 39 for detailed information about each function.

- **HPOM Configuration API**

  The HPOM Configuration API contains functions that let you work on HPOM objects rather than HPOM events. HPOM objects are, for example, nodes, node groups, message groups, applications, policies, and so on. It lets you set up a new HPOM environment, change the configuration of managed nodes, and modify message source policies.

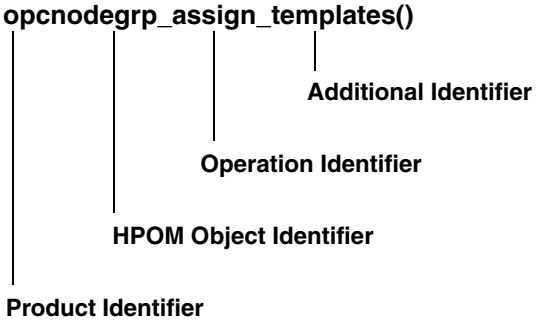
  See also Chapter 3, "Functions of the HPOM Configuration APIs," on page 175 for detailed information about each function.

See the *HPOM Application Integration Guide* for more information about the concept and use model of the APIs.

Function-Naming Conventions

The functions of the HPOM APIs have consistent names which reflect the operation they perform and the HPOM object on which they perform it. See Figure 1-1, “Naming the HPOM API Functions,” for an example of how the HPOM API functions are named.

Figure 1-1 Naming the HPOM API Functions



The function names consist of the following parts:

product identifier	Identifies the product, in HPOM this is always `opc`.
HPOM object identifier	Identifies the HPOM object on which the function performs the operation.
operation identifier	Identifies the operation which the function performs.
additional identifier	Additional description to identify what the function does or expects.

Table 1-1, “HPOM Objects,” on page 31 gives an overview of all available HPOM objects which can be manipulated with the APIs. The opcdata type must be used to describe the objects when using the APIs. See “HPOM Data Structures” on page 541 for more information about the opcdata types.

Table 1-1 HPOM Objects

HPOM Object	Description	opcdata Type
Action Request	Action request to start an action on a managed node. Used by the Legacy Link Interface.	OPCDTYPE_ACTION_REQUEST
Action Response	Action response from a previously started action on a managed node. Used by the Legacy Link Interface.	OPCDTYPE_ACTION_RESPONSE
Annotation	Message annotation.	OPCDTYPE_ANNOTATION
Application	Application used in HPOM	OPCDTYPE_APPLIC
Application Configuration	Configuration of an HPOM application. This object type is used to configure HPOM applications.	OPCDTYPE_APPL_CONFIG
Application Group	An application group is a container of applications and other application groups.	OPCDTYPE_APPL_GROUP
Application Response	An application response is the response of a previously started HPOM application. Application responses can be received using the Application Response Interface; see also “Interface API” on page 94.	OPCDTYPE_APPLIC_RESPONSE
Category Information	A category groups HPOM instrumentation files into a meaningful unit.	OPCDTYPE_CATEGORY_INFO
Configuration Event	A configuration event is sent when configuration was changed.	OPCDTYPE_CONFIG_EVENT

Table 1-1 HPOM Objects (Continued)

HPOM Object	Description	opcdata Type
Container	A container contains a list of objects of one type.	OPCDTYPE_CONTAINER
Instruction Text Interface	Configuration of an instruction text interface.	OPCDTYPE_INSTR_IF
Layout Group	A layout group contains a list of layout elements in a node hierarchy.	OPCDTYPE_LAYOUT_GROUP
Message	A message is the central management information element of the managed nodes.	OPCDTYPE_MESSAGE
Message Event	A message event is sent when a message was changed.	OPCDTYPE_MESSAGE_EVENT
Message Group	A message group is a grouping criteria of incoming messages.	OPCDTYPE_MESSAGE_GROUP
Message ID	A message ID contains the unique identifier of a message.	OPCDTYPE_MESSAGE_ID
Monitor Message	A monitor message is a monitor value which can be sent using the Agent Monitor API	OPCDTYPE_MONITOR_MESSAGE
Node	A node is an HP Operations managed node.	OPCDTYPE_NODE
Node Configuration	A node configuration is the configuration of an HP Operations managed node. It contains all necessary parameters to specify a node with all its characteristics.	OPCDTYPE_NODE_CONFIG
Node Group	A node group collects several nodes.	OPCDTYPE_NODE_GROUP
Node Hierarchy	A node hierarchy is a tree structure containing node layout elements and nodes as its leaves.	OPCDTYPE_NODEHIER

Table 1-1 HPOM Objects (Continued)

HPOM Object	Description	opcdata Type
Notification Schedule	A notification schedule defines the schedule for forwarding messages to a notification service.	OPCDTYPE_NOTI_SCHEDULE
Notification Service	A notification service configuration is an HPOM interface to a notification service.	OPCDTYPE_NOTI_SERVICE
Regroup Condition	A regroup condition regroups messages matching the specified condition.	OPCDTYPE_REGROUP_COND
Policy	A policy is used to configure message conditions on managed nodes.	OPCDTYPE_POLICY
Policy Group	A policy group collects several policies and other policy groups. Policy groups are handled like policies.	OPCDTYPE_POLICY_GROUP
Policy File	A policy file contains the complete configuration of a policy including its conditions. Policy files are only used by the policy file API.	[char *]
Policy Info	A policy info object contains the name, description, and type of a policy. It can be used to get a list of all available policy instead of the complete policy configuration.	OPCDTYPE_TEMPLATE_INFO
User Configuration	A user configuration contains the properties of an HPOM user.	OPCDTYPE_USER_CONFIG
User Profile	A user profile contains the properties of users and is assigned to users so that the user takes over the properties defined in the profile.	OPCDTYPE_USER_PROFILE

Libraries for HPOM Integrations

NOTE Customer applications must be linked to HPOM using the libraries and link and compile options given in the following tables. Integration is only supported if this is the case.

See the *HPOM HTTPS Agent Concepts and Configuration Guide* for the libraries and link and compile options for each supported managed node platform. The libraries and link and compile options for the management server are listed in the following section.

Libraries on the Management Server

Table 1-2 Libraries for the HPOM Management Server on HP-UX 11.31

Product Version	9.00
Library	libopcsv_r.so
Libraries linked to the HPOM library.	/opt/OV/lib/HPUX32/libopcdb.so /opt/OV/lib/HPUX32/libclntsh.so /opt/OV/lib/HPUX32/libnnz11.so /opt/OV/lib/HPUX32/libOvXpl.so /opt/OV/lib/HPUX32/libOvSecCore.so /opt/OV/lib/HPUX32/libOvBbc.so /opt/OV/lib/HPUX32/libOvCtrl.so /opt/OV/lib/HPUX32/libOvConf.so /opt/OV/lib/HPUX32/libOvDepl.so /opt/OV/lib/HPUX32/libOvOprEl.so /usr/lib/HPUX32/libpthread.so.1 /usr/lib/HPUX32/libnsl.so.1 /usr/lib/HPUX32/libpam.so.1 /usr/lib/HPUX32/libc.so.1 /usr/lib/HPUX32/librt.so.1 /usr/lib/HPUX32/libdl.so.1 /usr/lib/HPUX32/libm.so.1 /usr/lib/HPUX32/libIO77.so.1 /usr/lib/HPUX32/libunwind.so.1 /usr/lib/HPUX32/libuca.so.1
Link and compile options	-lopcsv_r -lopcdb -Inspsv -lpthread

NOTE Integrations compiled on HPOM 8.xx can run on the HPOM 9.00 management server without the need to re-compile or re-link. A re-link in necessary when you explicitly require the DCE library because your application directly or indirectly calls DCE functions and you have not linked the the DCE library. The HPOM libraries no longer provide the DCE library link.

Include Files and Makefiles

The appropriate HPOM include file on the management server is as follows:

`/opt/OV/include/opcapi.h`

See the *HPOM HTTPS Agent Concepts and Configuration Guide* for the location of the HPOM include files on all managed node platforms.

Examples of how API functions are used can be found in the `opcapitest.c` and `itoagtmsitest.c` files on the management server in the `/opt/OV/OpC/examples/copcagtapi` directory.

The `/opt/OV/OpC/examples/copcagtapi` directory on the management server also contains the makefiles for building the examples. They use the correct compile and link options needed to correctly build an executable.

The HPOM makefiles on the management server are as follows:

- HP-UX: `Makef.hpux11` and `Makef.hpuxIA32`

See the *HPOM HTTPS Agent Concepts and Configuration Guide* for the location of the HPOM makefiles on all managed node platforms and the makefile examples below:

- HP-UX: `/opt/OV/OpC/examples/progs/Makef.hpsv`

Using APIs in Internationalized Environments

All HPOM API functions are internationalized. This means that they will initialize the language setting, check the codeset for compatibility, and convert codesets if necessary, provided your API programs support Native Language Support (NLS) environments.

When writing API programs for internationalized environments, you must ensure that your programs do select the appropriate locale. In C programs, you do this by calling the function `setlocale()` at the beginning of your program.

It is recommended to use `setlocale(LC_ALL,"")`. The category `LC_ALL` names the program's entire locale. `""` adopts the setting of the current shell. As always, it is recommended to use a UTF-8 based locale.

Refer to the *setlocale(3C)* man page for more information about the `setlocale()` function.

HP Software Partnerships

The major benefit resulting from an integration with HPOM is the increased customer value of the integrated solution. HPOM is the industrial standard for problem management and supports a wide range of platforms which have either been developed internally, or by partners. When you integrate a solution with HPOM, it becomes part of a comprehensive management solution which meets customers' requirements for a unified system management approach. This increases the value your solution provides to customers, making it attractive to market segments that it couldn't previously address. A **partner program** has been established by Hewlett-Packard to support your integration efforts.

Integrations created by solution partners can be validated and certified by Hewlett-Packard to achieve the status of **HP Software Platinum Partner**. Validation ensures that the integration is well-behaved and does not conflict with other integrated solutions. As an HP Sofware Platinum Partner, your solution is recommended by HP sales channels, you can leverage from the well-established HP Software brand name, and you receive immediate market exposure for your solution through HP market awareness and selling tools.

For more information about the HP Software partner programs, see our web site at `http://h20229.www2.hp.com/partner/index.html`, and select `partners`.

2 Functions of the HPOM Operator APIs

In this Chapter

This chapter describes the functions of each of the following APIs:

- Data API
- Interface API
- Server Message API
- Agent Message API
- Agent Monitor API

NOTE

The files `opcapi.h` and `opcsvapi.h` contain predefined values for the function parameters, the function prototypes, and define the error codes. The files are located in: `/opt/OV/include/`.

Data API

The HPOM Data API provides a set of functions to set and get information in the form of HPOM data structures. Direct access to HPOM objects is not supported. This API is used for:

- HPOM Data Structures

  (See “HPOM Data Structures” on page 539 for more information.)

- Containers
- Iterators

Usage

To use the functions, include the header file `opcapi.h` or `opcsvapi.h` in your application.

Each routine returns an error/status code.

Prerequisites

The API functions can be issued by any user. For some attribute values, a maximum length applies as noted with the appropriate attribute selector described in the man page *opcdata(3)* and in “HPOM Data Structures” on page 539.

Multithread Usage

All functions of the HPOM Connection API are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

The HPOM Container

A **container** is used to contain an array of the type opcdata. The elements might contain messages, action requests, or any other type of opcdata, but not another container.

As the container contains copies of elements and not only references to elements, the memory of each element is allocated by the opcdata container function and is freed when an element is deleted or when the entire container is freed by opcdata_free().

To use the opcdata container functions, you must first create the container using:

```
opcdata_create (OPCDTYPE_CONTAINER, &container)
```

At this point the container is initialized and is ready to be filled with data; initially the container is of the type `OPCDTYPE_EMPTY`.

Before a program can be exited, the container must be freed using:

```
opcdata_free (&container)
```

This function deletes the container and its entire contents. All previously allocated memory is freed.

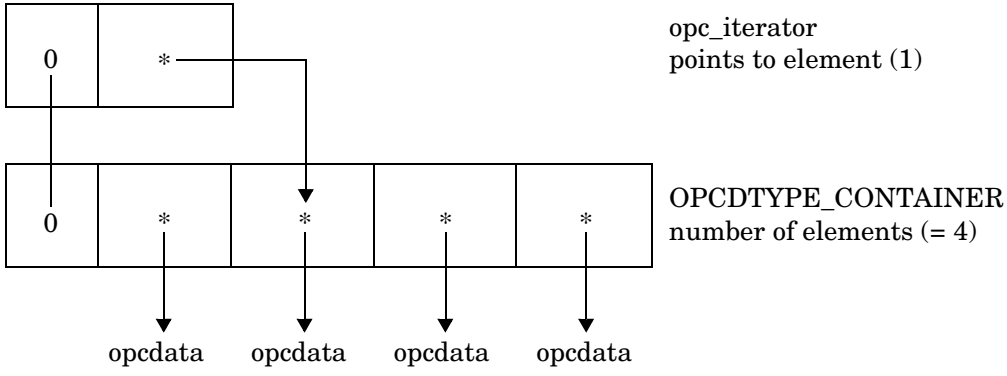
The HPOM Iterator

Looping through a container using the `opcdata_*` functions can be slow because a copy of the array element is returned. The copy must also be freed before the next element can be retrieved. The HPOM Iterator provides a set of routines that deal with references to opcdata elements instead of copies. These routines resemble the iterator mechanism used in object-oriented programming and the STL (Standard Template Library) in C++.

The Iterator functions step through a container and return references to data elements in the container. You can use these functions directly in the `opcdata_get`/`opcdata_set` function calls, to get or set information in an opcdata structure.

NOTE Do not free the memory of a returned referenced opcdata structure. This will corrupt the opcdata container.

Figure 2-1 Iterator and Container Structure



Before you can use the Iterator you must first create it using opciter_create(). This function allocates memory for the iterator structure and initializes it. Before a program can be exited, the allocated memory must be freed with opciter_free().

Some of these functions are not specifically iterator functions; instead, they enable you to set the iterator at a specified position in the container, or to get the current position of the iterator.

An iterator is always aligned with its container and can only be used with that container. To use an iterator with another container, you must delete the existing iterator and create a new one.

The container logs each iterator which is assigned to it. If the container is deleted, its iterators are set to invalid. This means that the pointer to the container in the iterator is set to NULL so that access to a non-existent container is not possible. This is controlled by the container function, described in the section “The HPOM Container” on page 42.

Registration Conditions of the HPOM Data API

The HPOM registration condition API is described in detail in “Registration Conditions of the HPOM Interface API” on page 98.

opcdata_append_element()

```
#include opcapi.h or opcsvapi.h

int opcdata_append_element (
       opcdata         container,              /* in */
       opcdata         element                 /* in */
                          );
```

Parameters

`container`	HPOM data structure of type OPCDTYPE_CONTAINER.
`element`	HPOM data structure.

Description

The function `opcdata_append_element()` appends a copy of the given element at the end of the container list. If `element` is the first element to be added to the container, it also specifies the type of the container. A container can contain only elements of one opcdata type.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`container` not of type OPCDTYPE_CONTAINER or NULL; `element` empty, NULL.

Versions

4.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_clear()

```
#include opcapi.h or opcsvapi.h

int opcdata_clear (
        opcdata * data      /*in/out*/
                  );
```

Parameters

`data` Points to the data area that will be cleared.

Description

Frees and re-initializes all fields in `data`.

NOTE This function changes the pointer to the data structure.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	Parameter `data` is invalid; probably NULL.
`OPC_ERR_CANT_INIT`	Unable to initialize.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

Versions

5.00 and later

opcdata_copy()

```
#include opcapi.h or opcsvapi.h

int opcdata_copy(
       const          opcdata data,          /* in */
       opcdata        *copy                  /* out */
                );
```

Parameters

data	HPOM data structure that will be copied.
copy	Copy of data.

Description

The API creates a copy of the data area and returns it in copy. It creates a complete copy, that is, the string fields of data are copied and not shared between data and copy. The allocated memory has to be deallocated using opcdata_free() before using this function. This function cannot be used to copy a data area of type OPCDTYPE_CONTAINER. If it is necessary to copy a whole container, the application must do this using iterator and container functions.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	data is NULL or of wrong type.
OPC_ERR_INVALID_OUTPARAM	copy is invalid; probably NULL.
OPC_ERR_NO_MEMORY	Memory allocation failed.

Versions

2.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_copy_info_to_actresp()

```
#include opcapi.h or opcsvapi.h

int opcdata_copy_info_to_actresp(
                const opcdata     data,       /* in */
                opcdata           copy        /* out */
                                  );
```

Parameters

`data`	HPOM data structure of type OPCDTYPE_ACTION_REQUEST or OPCDTYPE_MESSAGE.
`copy`	`copy` must be an opcdata structure of type OPCDTYPE_ACTION_RESPONSE.

Description

The function `opcdata_copy_info_to_actresp()` copies components from either a message or action request to an action response, depending on the type of `data`. If `data` points to a message, it is assumed that a response for a local automatic action is expected; if it points to an action request, all other kinds of actions are assumed except for local automatic actions.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`data` is NULL or of wrong type.
`OPC_ERR_INVALID_OUTPARAM`	`copy` is NULL.

Versions

2.00 and later

See Also

“HPOM Data Structures” on page 539

“OPCDTYPE_MONITOR_MESSAGE” on page 573

“OPCDTYPE_ACTION_REQUEST” on page 543

“OPCDTYPE_ACTION_RESPONSE” on page 545

opcdata_create()

```
#include opcapi.h or opcsvapi.h

int opcdata_create(
                   int          data_type,     /* in */
                   opcdata      *data          /* out */
                   );
```

Parameters

`data_type`	Specifies the type of the allocated data area; see “HPOM Data Structures” on page 539 for a list of supported data structures.
`data`	HPOM data structure.

Description

This function allocates and initializes a data structure. To get or set attributes, the respective routines must be called. The memory used for the area must be deallocated by calling opcdata_free().

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_OUTPARAM`	`data` is invalid; probably NULL.
`OPC_ERR_CANT_INIT`	Unable to initialize.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

Versions

2.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_delete_element()

```
#include opcapi.h or opcsvapi.h

int opcdata_delete_element (
                    opcdata    container,     /* in */
                    long       index          /* in */
                          );
```

Parameters

`container`	Pointer to an HPOM container of type OPCDTYPE_CONTAINER.
`index`	Position of the element within `container` to be deleted.

Description

The function `opcdata_delete_element()` deletes the element at the specified position in the container and also frees the memory of the removed element. The index must be in the interval:

```
0 < i < opcdata_num_elements(container)
```

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`container` not of type OPCDTYPE_CONTAINER or NULL.
`OPC_ERR_OUT_OF_RANGE`	`index` out of range (not in 0 … container_len-1).

Versions

3.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_free()

```
#include opcapi.h or opcsvapi.h

int opcdata_free(
              opcdata        *data        /* in/out */
                );
```

Parameters

`data`	Pointer to the data area that will be deallocated. `data` will be reset to NULL.

Description

The function `opcdata_free()` deallocates memory previously allocated by one of the functions opcif_read(), opcdata_create(), and opcdata_copy().

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`data` is NULL or of wrong type.

Versions

2.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_generate_id()

```
#include opcapi.h or opcsvapi.h

int opcdata_generate_id (
        opcdata   data       /* in */
       ,const int attribute  /* in */
                        );
```

Parameters

`data`	Pointer to the opcdata structure.
`attribute`	Specified which ID should be set.

Description

Generates an HPOM UUID and puts it into the ID field (specified in `attribute`) of the given opcdata structure.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	Input parameter not valid.

Versions

5.00 and later

opcdata_get_cma()

```
#include opcapi.h or opcsvapi.h

      int opcdata_get_cma(
         const    opcdata  data,          /* in */
         const    char     * name,        /* in */
         char              ** value       /* out */
                         );
```

Parameters

`data`	HPOM data structure of the type OPCDTYPE_MESSAGE.
`name`	Name of the attribute.
`value`	Value of the attribute.

Description

The function `opcdata_get_cma()` returns the value of the specified custom message attribute. A custom message attribute is specified by its `name`.

NOTE The memory allocated by this function must be freed by the caller.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	Input parameter `data` or `name` is invalid.
`OPC_ERR_INVALID_OUTPARAM`	Output parameter `value` is invalid; probably NULL.
`OPC_ERR_INVALID_OPCDATA_TYPE`	`data` is not of the type `OPCDTYPE_MESSAGE`.
`OPC_ERROR`	Unspecified problem.

Versions

7.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 573

“opcdata_get_cmanames()” on page 55

“opcdata_set_cma()” on page 73

“opcdata_remove_cma()” on page 71

opcdata_get_cmanames()

```
#include opcapi.h or opcsvapi.h

      int opcdata_get_cmanames(
         const    opcdata data,        /* in */
         char             *** names,   /* out */
         int              * len        /* out */
                             );
```

Parameters

`data`	HPOM data structure of the type OPCDTYPE_MESSAGE.
`names`	Returned pointer to the list of names.
`len`	Number of elements in the list.

Description

The function `opcdata_get_cmanames()` returns a list of the names of all available custom message attributes of a message. The caller must provide a pointer to a list of character pointers (`char ***`). The list will be returned to the caller.

NOTE The caller must free the memory of all elements in the list and the list itself.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	Input parameter `data` is invalid.
`OPC_ERR_INVALID_OUTPARAM`	Output parameter `names` or `len` is invalid; probably NULL.
`OPC_ERR_INVALID_OPCDATA_TYPE`	`data` is not of the type `OPCDTYPE_MESSAGE`.
`OPC_ERROR`	Problem cannot be specified.

Versions

7.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 573

“opcdata_get_cma()” on page 53

“opcdata_set_cma()” on page 73

“opcdata_remove_cma()” on page 71

opcdata_get_double()

```
#include opcapi.h or opcsvapi.h

double *opcdata_get_double(
          const          opcdata data,          /* in */
          int            attribute              /* in */
                           );
```

Parameters

`data`	HPOM data structure containing the queried attribute.
`attribute`	Specifies the attribute that is queried.

Description

The function `opcdata_get_double()` returns the value of a double attribute in `data`.

Return Values

Returns the real value, or, `OPC_DOUBLE_UNDEF` if no value could be retrieved because `data` was empty or `attribute` was not allowed. Note that the real value could also be `OPC_DOUBLE_UNDEF`.

If an error occurs, for example an invalid attribute is specified, the function returns `0.0`..

Versions

4.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 573

opcdata_get_element()

```
#include opcapi.h or opcsvapi.h

int opcdata_get_element (
            const opcdata    container,          /* in */
            opcdata          *element            /* out */
            long             index,              /* in */
                        );
```

Parameters

`container`	HPOM container of type OPCDTYPE_CONTAINER.
`element`	HPOM data structure pointer.
`index`	Position of the element in the container.

Description

The function `opcdata_get_element()` makes a copy of the specified element. Therefore, it allocates all necessary memory so that the user is responsible for its correct handling after use. This routine returns no references.

If `element` points to an existing HPOM data structure, it must be freed before calling this function. Otherwise the memory is lost because the pointer is overwritten by this function.

The index must be in the interval:

```
0 =< i < opcdata_num_elements(container)
```

Return Values

`int`	Number of elements.
`OPC_ERR_INVALID_INPARAM`	`container` not of type OPCDTYPE_CONTAINER or NULL.
`OPC_ERR_OUT_OF_RANGE`	`index` is out of range.

Versions

3.00 and later

See Also

“opciter_nth()” on page 84

“HPOM Data Structures” on page 539

opcdata_get_error_msg()

```
#include opcapi.h or opcsvapi.h

char* opcdata_get_error_msg (
        const int error_code        /* in  */
       ,char**    p_error_msg       /* out */
       ,int*      p_error_msg_size  /* out */
                          );
```

Parameters

`error_code` OPC_ERR_XXX error code.

`p_error_msg` Pointer to allocated error message string; memory must be freed by caller.

`p_error_msg_size` Size of allocated memory in bytes.

Description

Returns error text (description) relating to the given error code. The output is localized. This function is thread-safe.

NOTE The memory allocated by this function must be freed by the caller.

Return Values

string String contains error description.

Versions

5.00 and later

opcdata_get_long()

```
#include opcapi.h or opcsvapi.h

long opcdata_get_long(
          const       opcdata data,           /* in */
          int         attribute               /* in */
                      );
```

Parameters

`data`	Data structure containing the queried attribute.
`attribute`	Specifies the attribute that is queried.

Description

Returns the value of the numeric `attribute` in `data`.

Return Values

Returns the integer value of the attribute; if the routine fails, a value of -1 is returned.

Versions

2.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_get_str()

```
#include opcapi.h or opcsvapi.h

char *opcdata_get_str(
          const       opcdata data,          /* in */
          int         attribute              /* in */
                      );
```

Parameters

`data`	Data structure containing the queried attribute.
`attribute`	Specifies the attribute that is queried.

Description

Instead of a status value, the function `opcdata_get_str()` returns a pointer to the desired string value. This function can therefore be used directly in another function call.

Return Values

Returns a character pointer to the value of the defined attribute in the data area. The pointer points into the internal data area, and it can be invalidated by any subsequent API call issued against the same `opcdata` structure. You *must* use the API every time you want to look at a data item.

Modification of the attribute is only allowed using opcdata_set_str(); direct access to the string is not supported.

Versions

2.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_insert_element()

```
#include opcapi.h or opcsvapi.h

int opcdata_insert_element (
                opcdata          container,     /* in */
                opcdata          element,       /* in */
                long             index          /* in */
                          );
```

Parameters

`container`	HPOM data structure of type OPCDTYPE_CONTAINER.
`element`	HPOM data structure.
`index`	Position in the container where the element will be inserted.

Description

The function `opcdata_insert_element()` inserts a copy of the specified element at the specified position in the container. The same prerequisites apply as for opcdata_append_element(). The index must be in the interval:

```
0 =< i < opcdata_num_elements(container)
```

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`container` not of type OPCDTYPE_CONTAINER or NULL `element` empty, NULL or of type OPCDTYPE_CONTAINER.
`OPC_ERR_OUT_OF_RANGE`	`index` is out of range.

Versions

4.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“HPOM Data Structures” on page 539

opcdata_ladd()

```
#include opcapi.h or opcsvapi.h

long opcdata_ladd (
        const opcdata data,     /* in */
       ,const int     list      /* in */
                  );
```

Parameters

`data`	Pointer to an HPOM data structure with the embedded list.
`list`	Specifies the list in the data structure.

Description

Adds an element of the correct type to the specified list in the opcdata structure. The element type is specified with the list. After adding the element, the new length of the list will be returned.

Return Values

long	Long number of elements (0 ... list_len-1).
`OPC_ERR_INVALID_INPARAM`	Input parameter was not valid (< 0).

Versions

5.00 and later

opcdata_ldel()

```
#include opcapi.h or opcsvapi.h

long opcdata_ldel (
        const opcdata data      /* in */
       ,const int     list      /* in */
       ,const long    index     /* in */
                  );
```

Parameters

`data`	Pointer to an HPOM data structure with the embedded list.
`list`	Specifies the list in the data structure.
`index`	Specifies the element in the list.

Description

Deletes an element of the specified list in the opcdata structure. This function returns the new number of elements in the list.

Return Values

long	Long number of elements.
`OPC_ERR_INVALID_INPARAM`	Parameter `data` is invalid; probably NULL; `list` or `index` is invalid.

Versions

5.00 and later

opcdata_lget_len()

```
#include opcapi.h or opcsvapi.h

long opcdata_lget_len (
        const opcdata data     /* in */
       ,const int     list     /* in */
                      );
```

Parameters

`data`	Pointer to an HPOM data structure with the embedded list.
`list`	Specifies the list in the data structure.

Description

Returns the number of elements in an embedded list of an HPOM opcdata structure.

Possible data structures: `OPCDTYPE_APPL_CONFIG`.

Return Values

long	Long number of elements (0 ... maxint - 1).
`OPC_ERR_INVALID_INPARAM`	Parameter `data` is invalid; probably NULL; `list` is invalid.

Versions

5.00 and later

opcdata_lget_long()

```
#include opcapi.h or opcsvapi.h

long opcdata_lget_long (
        const opcdata data       /* in */
       ,const int     list       /* in */
       ,const int     attribute  /* in */
       ,const long    index      /* in */
                       );
```

Parameters

`data`	Pointer to an HPOM data structure with the embedded list.
`list`	Specifies the list in the data structure.
`attribute`	Specifies the attribute within the list element.
`index`	Index of the list element (0 ... list_len-1).

Description

Returns the value of the attribute in the list element.

Return Values

long	Value of the attribute.
`LONG_MIN`	An error occurred.

Versions

5.00 and later

opcdata_lget_str()

```
#include opcapi.h or opcsvapi.h

char * opcdata_lget_str (
        const opcdata data       /* in */
       ,const int     list       /* in */
       ,const int     attribute  /* in */
       ,const long    index      /* in */
                      );
```

Parameters

`data`	Pointer to an HPOM data structure with the embedded list.
`list`	Specifies the list in the data structure.
`attribute`	Specifies the attribute within the list element.
`index`	Index of the list element (0 ... list_len-1).

Description

Returns the pointer to the string in the attribute in the list element.

Return Values

`NULL`	An error occurred.

Versions

5.00 and later

opcdata_lset_long()

```
#include opcapi.h or opcsvapi.h

int opcdata_lset_long (
        const opcdata data       /* in */
       ,const int     list       /* in */
       ,const int     attribute  /* in */
       ,const long    index      /* in */
       ,const long    value      /* in */
                      );
```

Parameters

`data`	Pointer to an HPOM data structure with the embedded list.
`list`	Specifies the list in the data structure.
`attribute`	Specifies the attribute within the list element.
`index`	Index of the list element (0 ... list_len-1).
`value`	Long value for replacement.

Description

Replaces the value in the attribute of the list element.

Return Values

`OPC_ERROR_OK`	OK
`OPC_ERROR_OUT_OF_RANGE`	Index is out of range.
`OPC_ERR_INVALID_INPARAM`	Input parameter was not valid.

Versions

5.00 and later

opcdata_lset_str()

```
#include opcapi.h or opcsvapi.h

int opcdata_lset_str (
        const opcdata data       /* in */
       ,const int     list       /* in */
       ,const int     attribute  /* in */
       ,const long    index      /* in */
       ,const char *  value      /* in */
                     );
```

Parameters

`data`	Pointer to an HPOM data structure with the embedded list.
`list`	Specifies the list in the data structure.
`attribute`	Specifies the attribute within the list element.
`index`	Index of the list element (0 ... list_len-1).
`value`	String value for replacement.

Description

Replaces the string in the attribute of the list element.

Return Values

`OPC_ERROR_OK`	OK
`OPC_ERROR_OUT_OF_RANGE`	Index is out of range.
`OPC_ERR_INVALID_INPARAM`	Input parameter was not valid.

Versions

5.00 and later

opcdata_num_elements()

```
#include opcapi.h or opcsvapi.h

long opcdata_num_elements (
      const opcdata       container        /* in */
                         );
```

Parameters

`container`	HPOM data structure of type OPCDTYPE_CONTAINER.

Description

The function `opcdata_num_elements()` returns the number of elements of the specified container.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`container` not of type OPCDTYPE_CONTAINER or NULL.

Versions

4.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_remove_cma()

```
#include opcapi.h or opcsvapi.h

     int opcdata_remove_cma (
        const opcdata data,          /* in */
        const char * name            /* in */
                              );
```

Parameters

`data`	Data structure of the type OPCDTYPE_MESSAGE.
`name`	Name of the attribute to be removed.

Description

`opcdata_remove_cma()` removes the specified custom message attribute from a message. If the specified custom message attribute does not exist in a message, OK will be returned.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	Parameter `data` or `name` is not valid.
`OPC_ERR_INVALID_OPCDATA_TYPE`	`data` is not of type `OPCDTYPE_MESSAGE`.
`OPC_ERROR`	Problem cannot be specified.

Versions

7.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 573

“opcdata_get_cma()” on page 53

“opcdata_get_cmanames()” on page 55

“opcdata_set_cma()” on page 73

opcdata_report_error()

```
#include opcapi.h or opcsvapi.h

char * opcdata_report_error (
        const int error_code      /* in */
                                  );
```

Parameters

`error_code` OPC_ERR_XXX error code.

Description

Returns the error text (description) relating to the given error code.

NOTE The memory allocated by this function must be freed by the caller.

Return Values

string String with error description.

Versions

5.00 and later

opcdata_set_cma()

```
#include opcapi.h or opcsvapi.h

     int opcdata_set_cma (
        const opcdata   data,          /* in */
        const char      * name,        /* in */
        const char      * value        /* in */
                         );
```

Parameters

`data`	Data structure of the type OPCDTYPE_MESSAGE.
`name`	Name of the attribute to be set.
`value`	Value of the attribute to be set.

Description

This function sets a custom message attribute with the specified name and value in a message. If a custom message attribute with the specified name exists already, it will be replaced; otherwise a new attribute will be added to the message.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	Parameter `data`, `name`, or `value` is invalid.
`OPC_ERR_INVALID_OPCDATA_TYPE`	`data` is not of type `OPCDTYPE_MESSAGE`.

Versions

7.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 573

“opcdata_get_cma()” on page 53

“opcdata_get_cmanames()” on page 55

“opcdata_remove_cma()” on page 71

opcdata_set_double()

```
#include opcapi.h or opcsvapi.h

int *opcdata_set_double(
       opcdata          data,                 /* in/out */
       int              attribute,            /* in */
       double           value                 /* in */
                        );
```

Parameters

`data`	HPOM data structure containing the attribute to be set
`attribute`	Specifies the attribute.
`value`	Contains the value of the attribute to be set.

Description

The function `opcdata_set_double()` sets the numeric float `attribute` in `data` to `value`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`data` is NULL, `attribute` is NULL, `value` is NULL.

Versions

4.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_set_long()

```
#include opcapi.h or opcsvapi.h

int opcdata_set_long(
                  opcdata      data,         /* in/out */
                  int          attribute,    /* in */
                  long         value         /* in */
                     );
```

Parameters

`data`	HPOM data structure containing the attribute to be set.
`attribute`	Specifies the attribute.
`value`	Contains the value of the attribute to be set.

Description

Use the `opcdata_set_long()` routine to set the numeric long `attribute` in `data` to `value`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`data` is NULL, `attribute` is invalid.

Versions

2.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_set_str()

```
#include opcapi.h or opcsvapi.h

int *opcdata_set_str(
        opcdata         data,               /* in */
        int             attribute,          /* in */
        const char      *value              /* in */
                    );
```

Parameters

`data`	HPOM data structure.
`attribute`	Select the attribute that is set.
`value`	Specify the value of the attribute.

Description

Use the function opcdata_set_str() to set the string `attribute` to a copy of `value`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`data` is NULL, `attribute` is invalid, `value` is NULL.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

Versions

2.00 and later

See Also

“HPOM Data Structures” on page 539

opcdata_type()

```
#include opcapi.h or opcsvapi.h

int opcdata_type (
        opcdata data  /* in */
                 );
```

Parameters

`data`	Points to an initialized data area.

Description

Returns the opcdata type in `data`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`data` is invalid; probably NULL.

Versions

5.00 and later

opciter_begin()

```
#include opcapi.h or opcsvapi.h

opcdata opciter_begin (
      opc_iterator            iterator         /* in */
                        );
```

Parameters

`iterator` HPOM Iterator.

Description

The function `opciter_begin()` returns the pointer of the first element of its container and sets the iterator to the first position (pos = 0).

Return Values

Returns a reference of the type opcdata. If the container is empty, or an error occurred, this function returns NULL.

Versions

4.00 and later

See Also

“The HPOM Iterator” on page 43

opciter_create()

```
#include opcapi.h or opcsvapi.h

int opciter_create (
        opc_iterator                *iterator,  /* out */
        opcdata                      container  /* in */
                    );
```

Parameters

iterator	HPOM Iterator.
container	HPOM container of type OPCDTYPE_CONTAINER.

Description

The function opciter_create() creates an iterator for the specified container. The iterator will be returned in the given pointer to opc_iterator. The function opciter_create() allocates memory for the iterator structure, initializes it and appends its address to the iterator list of its container.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	An input parameter was invalid.
OPC_ERR_INVALID_OUTPARM	The output parameter was invalid.
OPC_ERR_NO_MEMORY	Unable to allocate memory.

Versions

4.00 and later

See Also

“The HPOM Iterator” on page 43

“HPOM Data Structures” on page 539

opciter_end()

```
#include opcapi.h or opcsvapi.h

opcdata opciter_end (
          opc_iterator          iterator          /* in */
                    );
```

Parameters

`iterator` HPOM Iterator.

Description

The function `opciter_end()` returns the pointer to the past-end-element. Because the past-end-element does not exist, it returns a NULL pointer. The position of the iterator will not be changed by this function. This function is useful as break condition in loops, and to test if the last element was reached.

Return Values

Returns a reference of the type opcdata.

NULL Past-end-value.

Versions

4.00 and later

See Also

“The HPOM Iterator” on page 43

“HPOM Data Structures” on page 539

opciter_free()

```
#include opcapi.h or opcsvapi.h

int opciter_free (
       opc_iterator          *iterator       /* in/out */
                  );
```

Parameters

`iterator` HPOM Iterator.

Description

The function `opciter_free()` frees previously allocated memory from the given iterator. It also deletes its pointer from the iterator list of its container.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_INPARAM`	An input parameter was invalid.

Versions

4.00 and later

See Also

“The HPOM Iterator” on page 43

“HPOM Data Structures” on page 539

opciter_get_pos()

```
#include opcapi.h or opcsvapi.h

long opciter_get_pos (
        opc_iterator         iterator            /* in */
                    );
```

Parameters

`iterator` HPOM Iterator.

Description

The function `opciter_get_pos()` returns the index of the container element to which the iterator points.

Return Values

A long value in the range `[ 0 <= retval < container_len ]` or `OPC_ERR_INVALID_INPARAM` in case of an invalid inparameter.

Versions

4.00 and later

See Also

“The HPOM Iterator” on page 43

“HPOM Data Structures” on page 539

opciter_next()

```
#include opcapi.h or opcsvapi.h

opcdata opciter_next (
         opc_iterator            iterator       /* in */
                     );
```

Parameters

`iterator` HPOM Iterator.

Description

The function `opciter_next()` returns the pointer of the next element (element[pos+1]) and increments the iterator by one (pos++). When the last element is reached, `opciter_next()` returns NULL (the past-end-element). If the container is empty, NULL will also be returned.

Return Values

Returns a reference of the type opcdata to the next element, or NULL if the call was not successful.

Versions

4.00 and later

See Also

“The HPOM Iterator” on page 43

“HPOM Data Structures” on page 539

opciter_nth()

```
#include opcapi.h or opcsvapi.h

opcdata opciter_nth (
         opc_iterator      iterator,            /* in */
         long              index                /* in */
                    );
```

Parameters

`iterator`	HPOM Iterator.
`index`	Position of the element.

Description

The function `opciter_nth()` returns the reference to the given container element. The iterator itself will be left unchanged. This means that it will not be set to the given element. If the container is empty or the given index is not in the range `0 < index < container_len`, then the `opciter_nth()` function returns NULL.

Note that whereas `opciter_nth()` only returns the reference to the element in the container, opcdata_get_element() returns a copy of the element.

Return Values

Returns a reference of the type opcdata to the specified element or NULL if the call was not successful.

Versions

4.00 and later

See Also

“opcdata_get_element()” on page 58

“HPOM Data Structures” on page 539

opciter_prev()

```
#include opcapi.h or opcsvapi.h

opcdata opciter_prev (
         opc_iterator          iterator         /* in */
                     );
```

Parameters

`iterator` HPOM Iterator.

Description

The function `opciter_prev()` returns the pointer of the previous element in the container (element[pos-1]) and decrements the iterator by one (pos--). When the first element is reached, or the container is empty, the iterator will be not further decremented and returns NULL.

Return Values

Returns a reference of the type opcdata to the previous element or NULL if the call was not successful.

Versions

4.00 and later

See Also

“HPOM Data Structures” on page 539

opciter_set_pos()

```
#include opcapi.h or opcsvapi.h

int  opciter_set_pos (
       opc_iterator              iterator,          /* in */
       long                      index              /* in */
                        );
```

Parameters

`iterator`	HPOM Iterator.
`index`	Position in the container.

Description

The function `opciter_set_pos()` sets the iterator to the specified position in the container. The index must be in the range `0 < index < container_len`.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_INPARAM`	The iterator is not valid.
`OPC_ERR_OUT_OF_RANGE`	The index is out of range.

Versions

4.00 and later

See Also

“opciter_nth()” on page 84

“HPOM Data Structures” on page 539

opcreg_copy()

```
#include opcapi.h or opcsvapi.h

int   opcreg_copy (
      const opcregcond     reg_cond,       /* in */
      opcregcond           *copy           /* out */
                  );
```

Parameters

reg_cond	Pointer to the registration condition to copy.
copy	Address of the pointer to the copied condition.

Description

The function opcreg_copy() creates a complete copy of a registration condition and returns it in copy. The allocated memory has to be deallocated using opcreg_free().

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_NO_MEMORY	Memory allocation error.
OPC_ERR_INVALID_OUTPARAM	copy is NULL or not of the correct type.

Versions

2.00 and later

See Also

“opcreg_create()” on page 88

“opcreg_free()” on page 89

“opcreg_get_long()” on page 90

“opcreg_get_str()” on page 91

“opcreg_set_long()” on page 92

“opcreg_set_str()” on page 93

opcreg_create()

```
#include opcapi.h or opcsvapi.h

int   opcreg_create (
      opcregcond          *reg_cond          /* out */
                    );
```

Parameters

`reg_cond` Address of the pointer to the registration condition.

Description

The function `opcreg_create()` allocates and initializes an empty registration condition and returns a pointer to the created structure. The actual structure of this data area is hidden from the user. To get or set attributes, the respective routines must be called. The memory used for the area has to be deallocated by calling `opcreg_free()`.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_OUTPARAM`	`reg_cond` is NULL or not of the correct type.
`OPC_ERR_NO_MEMORY`	Memory allocation error.

Versions

2.00 and later

See Also

“opcreg_free()” on page 89

“opcreg_copy()” on page 87

“opcreg_get_long()” on page 90

“opcreg_get_str()” on page 91

“opcreg_set_long()” on page 92

“opcreg_set_str()” on page 93

opcreg_free()

```
#include opcapi.h or opcsvapi.h

int   opcreg_free (
      opcregcond          *reg_cond          /* in/out */
                  )
```

Parameters

`reg_cond`	Address of the pointer to the registration condition.

Description

The function `opcreg_free()` deallocates the memory associated with a registration condition.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_OUTPARAM`	`reg_cond` is NULL or not of the correct type.

Versions

2.00 and later

See Also

“opcreg_create()” on page 88

“opcreg_copy()” on page 87

“opcreg_get_long()” on page 90

“opcreg_get_str()” on page 91

“opcreg_set_long()” on page 92

“opcreg_set_str()” on page 93

opcreg_get_long()

```
#include opcapi.h or opcsvapi.h

int   opcreg_get_long (
      const opcregcond     reg_cond,       /* in */
      int                  field           /* in */
                    );
```

Parameters

`reg_cond`	Registration condition containing the numeric value.
`field`	Specifies the attribute that is queried.

Description

Use the function `opcreg_get_long()` to access the attribute values of a condition.

Return Values

Returns the requested long value, or, if not successful -1.

Versions

2.00 and later

See Also

“opcreg_create()” on page 88

“opcreg_free()” on page 89

“opcreg_copy()” on page 87

“opcreg_get_str()” on page 91

“opcreg_set_long()” on page 92

“opcreg_set_str()” on page 93

opcreg_get_str()

```
#include opcapi.h or opcsvapi.h

int   *opcreg_get_str (
      const opcregcond     reg_cond,       /* in */
      int                  field           /* in */
                      );
```

Parameters

`reg_cond`	Registration condition containing the string.
`field`	Specifies the attribute that is queried.

Description

Use the function `opcreg_get_str()` to access the string attribute of a registration condition.

Return Values

Returns a character pointer to the value of the defined attribute in the data area. The pointer points into the internal data area. Modification of the attribute is only allowed using opcreg_set_str(); direct access to the string is not supported, however, it is not possible to prevent the user from committing direct modifications. If not successful a NULL pointer is returned.

Versions

2.00 and later

See Also

“opcreg_create()” on page 88

“opcreg_free()” on page 89

“opcreg_copy()” on page 87

“opcreg_get_long()” on page 90

“opcreg_set_long()” on page 92

“opcreg_set_str()” on page 93

opcreg_set_long()

```
#include opcapi.h or opcsvapi.h

int   opcreg_set_long (
      const opcregcond     reg_cond,       /* in/out */
      int                  field,          /* in */
      long                 value           /* in */
                      );
```

Parameters

`reg_cond`	Registration condition containing the string.
`field`	Selects the attribute in the registration condition.
`value`	Specifies the value of the attribute in the registration condition.

Description

Use the function `opcreg_set_long()` to set the value of a numeric attribute of a registration condition.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_FIELD`	Invalid value used for `field`.

Versions

2.00 and later

See Also

“opcreg_create()” on page 88

“opcreg_free()” on page 89

“opcreg_copy()” on page 87

“opcreg_get_long()” on page 90

“opcreg_get_str()” on page 91

“opcreg_set_str()” on page 93

opcreg_set_str()

```
#include opcapi.h or opcsvapi.h

int   *opcreg_set_str (
      const opcregcond     reg_cond,       /* in/out */
      int                  field,          /* in */
      const char           *value          /* in */
                        );
```

Parameters

`reg_cond`	Registration condition containing the string.
`field`	Selects the attribute in the registration condition.
`value`	Specifies the value of the attribute in the registration condition.

Description

The function `opcreg_set_str()` sets the value of a string attribute of a registration condition.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_FIELD`	Invalid value used for `field`.

Versions

2.00 and later

See Also

“opcreg_create()” on page 88

“opcreg_free()” on page 89

“opcreg_copy()” on page 87

“opcreg_get_long()” on page 90

“opcreg_get_str()” on page 91

“opcreg_set_long()” on page 92

Interface API

HPOM Interfaces

This API provides access to the HPOM Interfaces. These interfaces are designed to provide access to HPOM management information and are separated into three interfaces:

- ❏ “Message Stream Interface” on page 94
- ❏ “Configuration Stream Interface” on page 96
- ❏ “Message Event Interface” on page 97
- ❏ “Application Response Interface” on page 97

Message Stream Interface

This interface makes it possible to read HPOM messages from and to write messages into the internal message stream. The Message Stream Interface (MSI) is available on the HP Operations management server and on HPOM managed modes. All MSI types establish a connection, either to the HPOM message manager for server types (`OPCSVIF_*`) or to the HPOM message agent for agent types (`OPCAGTIF_*`).

Interface Types on the Management Server	Interface Types on the Managed Nodes
`OPCSVIF_EXTMSGPROC_READ` This interface type is used for non-destructive read operations on the HPOM internal message flow. Only messages that are allowed to be output on the Server MSI are accessible via this interface type. This type of interface is typically used by statistical analysis tools or additional display facilities.	`OPCAGTIF_EXTMSGPROC_READ` As `OPCSVIF_EXTMSGPROC_READ` but for the Agent MSI.

Interface Types on the Management Server	Interface Types on the Managed Nodes
OPCSVIF_EXTMSGPROC_READWRITE This interface type is used to read messages from HPOM's internal message flow, to modify / create messages, and to write them back to the HPOM processes. Only messages which are allowed to be output on the MSI are accessible via this interface type. Messages tagged with 'copy to' remain in HPOM's message flow, whereas messages tagged with 'divert to' are taken out of the flow. This type of interface could be used by event correlation engines.	OPCAGTIF_EXTMSGPROC_READWRITE As OPCSVIF_EXTMSGPROC_READWRITE but for the Agent MSI.
OPCSVIF_EXTMSGPROC_WRITE Interface instances of this type are used for write-only applications, to feed messages into HPOM's message flow. This type of interface could also be used to encapsulate the opcif_write() routine in a command line interface.	OPCAGTIF_EXTMSGPROC_WRITE As OPCSVIF_EXTMSGPROC_WRITE for the Agent MSI.
OPCSVIF_EXTAGT_MESSAGE This interface type is used to write messages to the HPOM message flow but is only available on the HP Operations management server. Together with interface instances of the types OPCSVIF_EXTAGT_ACTION_REQUEST and OPCSVIF_EXTAGT_ACTION_RESPONSE, it can be used to plug in custom-made external agents. Messages which are written to an interface instance of this type are still accessible via OPCSVIF_EXTMSGPROC_READ and OPCSVIF_EXTMSGPROC_READWRITE interface instances.	Not available.
OPCSVIF_EXTAGT_ACTION_REQUEST External agents use this interface type to read action requests.	Not available.

Interface Types on the Management Server	Interface Types on the Managed Nodes
OPCSVIF_EXTAGT_ACTION_RESPONSE Interface instances of this type are used by external agents to write action responses to HPOM.	Not available.

Configuration Stream Interface

Configuration Stream Interface (CSI) is an extension of the Message Stream Interface (MSI) for synchronizing the configuration changes. CSI provides registration for the configuration changes to the internal (server processes, Java GUI) and external (API clients) configuration consumers. Java GUI and server processes are registered for the configuration changes by default.

Interface Types on the Management Server	Interface Types on the Managed Nodes
OPCSVIF_CFG_CHG_EVENTS This interface type enables registration for all configuration changes. Clients receive configuration change notifications regardless of the registered user. This interface type is mostly used for configuration synchronization of internal processes.	Not available.
OPCSVIF_CFG_CHG_EVENTS_GUI This interface type is appropriate for clients that register with a user name and would like to receive only events related to that operator. If a user is not entitled to see a change, the configuration change notification is not sent to this client. Such interface type is used with Java GUI.	Not available.

Message Event Interface

This interface allows receipt of HPOM message events. These events are always sent when one operator or the administrator changes the status of a message. This interface is available only on the HP Operations management server and establishes a connection to the display manager.

Interface Types on the Management Server	Interface Types on the Managed Nodes
`OPCSVIF_MSG_EVENTS` This interface type is used to receive message events caused by changes to messages by other operators.	Not available.
`OPCSVIF_MSG_EVENTS_ALL` This interface type is used to receive Message Events caused by changes to messages by other operators, in addition to new messages and a few formerly internal events (like duplicate received).	Not available.

Application Response Interface

This interface allows receipt of the textual response from previously started application. This interface is available only on the HP Operations management server and establishes a connection to the display manager.

Interface Types on the Management Server	Interface Types on the Managed Nodes
`OPCSVIF_APPLIC_RESPONSE` This interface type is used to receive Application Responses. These applications must be started via `opcappl_start()`. Other application responses are not accessible with this interface.	Not available.

Prerequisites

The API functions must be issued by the user root.

Multi-thread Usage

All functions of the HPOM Connection API are safe to be called by multi-threaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

`opcreg_copy()` is not thread-safe for POSIX threads.

Registration Conditions of the HPOM Interface API

HPOM provides a user-accessible data type to define registration conditions as the mechanism to register with the HPOM Interfaces.

The registration condition API provides a set of functions to access the attributes in HPOM registration condition structure. The registration conditions are the conditions for the filtering of incoming data in the HPOM interfaces (see also opcif_api(3)). These functions are needed to create an empty condition, modify or query condition fields, to duplicate or delete a condition definition from memory, to set attributes to a value or get the value of an attribute to define filter conditions in the HPOM interfaces.

Memory for the configuration data is allocated on the heap.

Security Considerations for the HPOM Interface API

As an event subscription service API is a window for applications to see generic system-wide activity, applications must be prevented from unauthorized snooping of system behavior at this access point. In addition, access to the HPOM message flow in read-write mode allows an external application to discard messages, without a user being made aware that a message was generated. The APIs must, therefore, apply authentication mechanisms to prevent users and applications from unauthorized access to the HPOM message flow.

Automatic and Operator-initiated Actions

One important and critical issue arising from these security considerations is whether external applications using the interfaces are allowed to define automatic and/or operator-initiated actions. If HPOM allows access to these message attributes, any user who is authorized to call the APIs is also able to execute actions on HP Operations managed nodes.

According to the current HPOM concept, which regards HPOM as an open application providing a high level of flexibility to integrate applications, HPOM allows external programs to define actions for messages that are passed to the message manager. Event correlation can be seen as an advance on the existing concept of message conditions (“if attributes match then set attributes and actions”) to a higher level (“if rule fires then set attributes and actions”). It is, therefore, essential that these external applications are allowed to perform these modifications. An appropriate authorization mechanism at the API level guarantees that only authorized users can apply the APIs.

The HPOM GUI provides a possibility to disable the interface functionality. In addition, you can configure whether actions can be defined by an application that is writing to the interface. This concerns all interface types except `OPCSVIF_MSG_EVENTS` and `OPCSVOF_APPLIC_RESPONSE` which are currently read-only interfaces.

You can also define whether each message is allowed for output to the Message Stream Interface in the HPOM Administrator’s GUI. For example, an administrator can prevent the output of certain messages so that external applications do not receive secure information by reading these messages from the HPOM message flow.

Summary

HPOM allows users with a user ID of zero (uid 0), typically root, to access the HPOM Interface APIs and to define actions for messages that are sent to the management server. In the GUI, the HPOM administrator can enable or disable the interface functionality of the interface types concerning the message flow and allow or disallow actions that are read from the interface. If actions are disallowed, an appropriate error text is added to the annotations field and the action disabled. Access to Message Events is not configurable because it is not possible to manipulate message by way of this interface.

opcif_close()

```
#include opcapi.h or opcsvapi.h

int opcif_close(
                  int         interface_id       /* in */
                );
```

Parameters

`interface_id` Specifies which interface instance is used.

Description

Call the `opcif_close()` API to terminate the connection to the interface and disconnect from the HPOM server processes.

If the interface is opened with `OPCIF_CLOSE_DISCARD` specified in the mode parameter in the opcif_open() call, all unread data currently in the interface queue will be discarded, that is, messages diverted into the interface will be lost.

If the interface is opened in read/write mode and `OPCIF_CLOSE_FORWARD` is specified in the opcif_open() call, all data in the interface queue is forwarded before the queue is removed; if not, data in the input queue is discarded.

If the interface is opened with `OPCIF_BUFFER`, then the interface will be closed and *no* buffering will take place. Thus, this routine is used to shut down an interface. If buffering should take place during application downtime, use opcif_close_and_buffer() instead.

Message events and application responses are always discarded when the interface instance is closed.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_CANT_INIT`	Initialization of queues failed.
`OPC_ERR_INVALID_INTERFACE_ID`	No such interface opened.

Versions

2.00 and later

See Also

“HPOM Interfaces” on page 94

“Data API” on page 41

“HPOM Data Structures” on page 539

opcif_close_and_buffer()

```
#include opcapi.h or opcsvapi.h

int opcif_close_and_buffer(
                  int         interface_id       /* in */
                 );
```

Parameters

`interface_id` Specifies which interface instance is used.

Description

The `opcif_close_and_buffer()` API closes an open interface instance, and buffers all messages or message change events that are registered for this interface after closing it. The interface must be opened in the `OPCIF_BUFFER` mode, otherwise the error `OPC_ERR_INVALID_INTERFACE_TYPE` will be raised.

Note that this API is intended for applications that are often restarted but should still be able to process messages and message change events that occur during application downtime. If an application is never restarted (and the buffered data is not read), the queue will grow and is only limited by file system size or the `max_entries` parameter, if any.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_CANT_INIT`	Initialization of queues failed.
`OPC_ERR_INVALID_INTERFACE_ID`	No such interface opened.

Versions

7.18 and later

See Also

“HPOM Interfaces” on page 94

“Data API” on page 41

“HPOM Data Structures” on page 539

opcif_get_pipe()

```
#include opcapi.h or opcsvapi.h

int opcif_get_pipe(
                  int       interface_id,       /* in */
                  int       pipefd              /* out */
                   );
```

Parameters

interface_id	Specifies which interface instance is used.
pipefd	Returns the file descriptor of the selected output interface queue.

Description

A program reading from several input files needs the pipe file descriptor of its interface input queue. This descriptor is then used as part of the parameters to `select(2)` or `fcntl(2)`. The function `opcif_get_pipe()` returns this value.

For convenience reasons, an application that reads only from the HPOM interface can specify `OPCIF_READ_WAIT` in the opcif_open() call.

If the interface is opened with `OPCIF_BUFFER` and the interface is currently in synchronization mode, that is, the buffered elements are being delivered to the application, then this routine returns `OPC_ERR_OK` with the pipe file descriptor -1. In this case the buffered elements must be read first using the API opcif_read() until the return code `OPC_ERR_NO_DATA` is encountered. This code signals that synchronization mode is finished.

The application may then call `opcif_get_pipe()` again and a valid pipe file descriptor will be returned which may be then used in a select() call.

Return Values

OPC_ERR_OK	OK
OPC_ERR_CANT_INIT	Initialization of queues failed.
OPC_ERR_INVALID_OUTPARAM	`pipefd` is NULL.
OPC_ERR_INVALID_INTERFACE_ID	No such interface opened.

Versions

2.00 and later

See Also

“HPOM Interfaces” on page 94

opcif_open()

```
#include opcapi.h or opcsvapi.h

int opcif_open(
                int          interface_type,     /* in */
                const char   *instance,          /* in */
                int          mode,               /* in */
                int          max_entries,        /* in */
                int          *interface_id       /* out */
              );
```

Parameters

`interface_type` Specifies the type of interface to use from:

Server Message Stream Interface

Used by external message processors (for example, event correlation engines):

- `OPCSVIF_EXTMSGPROC_READ`
- `OPCSVIF_EXTMSGPROC_READWRITE`
- `OPCSVIF_EXTMSGPROC_WRITE`

Agent Message Stream Interface

Used by external message processors (for example, event correlation engines):

- `OPCAGTIF_EXTMSGPROC_READ`
- `OPCAGTIF_EXTMSGPROC_READWRITE`
- `OPCAGTIF_EXTMSGPROC_WRITE`

Configuration Stream Interface

Used by server and GUI processes:

- `OPCSVIF_CFG_CHG_EVENTS`
- `OPCSVIF_CFG_CHG_EVENTS_GUI`

Legacy Link Interface

Used by external message tools which work like an agent:

- `OPCSVIF_EXTAGT_MESSAGE` (write message)

- `OPCSVIF_EXTAGT_ACTION_REQUEST` (get request)
- `OPCSVIF_EXTAGT_ACTION_RESPONSE` (send response)

Message Event Interface

Used by external message logging and processing tools (for example, message browsers)

- `OPCSVIF_MSG_EVENTS`
- `OPCSVIF_MSG_EVENTS_ALL`

Application Response Interface

Used by external application processing tools:

- `OPCSVIF_APPLIC_RESPONSE`

`instance` User-defined name of the interface instance which is registered for one of the interface types above. The name is limited to a length of 13 alpha-numeric characters.

It is not possible to open several interface instances with the same instance name. This causes undefined behavior, especially when the mode parameter `OPCIF_IGNORE_MSI_ALREADY_EXISTS` is used.

`mode` To specify whether the interface is opened if the HPOM processes are not running. Use either:

- `OPCIF_ALWAYS` (default)

  The interface is opened regardless of whether or not the HP Operations server or agent processes are running.

- `OPCIF_SV_RUNNING`

  The interface is only opened when the HP Operations server processes are running.

- `OPCIF_AGT_RUNNING`

  The interface is only opened when the HP Operations agent processes are running.

The following options specify whether the `opcif_read()` API will wait for available data or not; in the WAIT case, the calling process will be blocked until data is available or the process receives an interrupt:

- `OPCIF_READ_WAIT` (default)

  The calling process is blocked until data is available or until the process receives a signal.

- `OPCIF_READ_NOWAIT`

  opcif_read() returns with an appropriate error code if no data is available to read.

To specify the handling of unread messages if the connected process closes the interface or aborts, use one of the following options:

- `OPCIF_CLOSE_FORWARD` (default)

  All unread data diverted to the interface is fed back in the HPOM interface when the interface is closed or the connected process aborts.

- `OPCIF_CLOSE_DISCARD`

  All unread data diverted to the interface will be lost. This option is only valid for the `OPCSVIF_EXTMSGPROC_READWRITE`, `OPCAGTIF_EXTMSGPROC_READWRITE`, `OPCSVIF_MSG_EVENTS` or `OPCSVIF_APPLIC_RESPONSE` interface type.

The following options specifies how to buffer data:

- `OPCIF_BUFFER`

  All data is buffered if the application aborts or closes the interface with opcif_close_and_buffer(). This option is only valid for the `OPCSVIF_EXTMSGPROC_READWRITE`, `OPCSVIF_EXTMSGPROC_READ` or `OPCSVIF_MSG_EVENTS` interface type.

If buffering is activated on an interface for which the `max_entries` parameter is set to 0 (unlimited queue length), and messages are diverted to the interface, a situation may occur where internal critical messages may be lost in the queue. This may happen if the associated application aborts or terminates with buffering mode turned on. See opcif_read(), opcif_close(), and opcif_close_and_buffer() for details.

The following option lets you reuse an existing queue file.

- `OPCIF_IGNORE_MSI_ALREADY_EXISTS`

  Be careful when using this flag because the queue file could already be in use by another application.

It is possible to combine these options using the OR logical operator (`|`).

`max_entries` Specifies the maximum number of entries in the read-queues. If this number is exceeded, HPOM stops writing to the queue. When the reading process has emptied the queue, it is notified by an error value returned by `opcif_read()`. The application must then disconnect and reopen the interface. To disable this check, specify 0 for `max_entries`.

`interface_id` The returned value must be used in subsequent calls to the APIs to refer to this instance of the interface.

Description

Use the function `opcif_open()` to connect to an instance of one of the following interfaces:

- Server Message Stream Interface
- Agent Message Stream Interface
- Legacy Link Interface
- Message Event Interface
- Application Response Interface

`opcif_open()` creates and opens the related files and establishes a connection to the HPOM processes.

Make sure that all opened interface instances are closed correctly when the application exits or an interface instance is re-opened. Otherwise duplicate data or critical internal messages may appear in the message browser.

Return Values

`OPC_ERR_OK`	Interface correctly opened.
`OPC_ERR_INVALID_OUTPARAM`	Pointer to `interface_id` is invalid.
`OPC_ERR_ACCESS_DENIED`	Access denied; root access is necessary.
`OPC_ERR_INVALID_INTERFACE_INSTANCE`	Instance name contains invalid characters, or is too long.
`OPC_ERR_INVALID_INTERFACE_TYPE`	No such interface type.
`OPC_ERR_CANT_INIT`	Initialization of queues failed.
`OPC_ERR_CANT_OPEN_READQUEUE`	Unable to open readqueue.
`OPC_ERR_CANT_OPEN_WRITEQUEUE`	Unable to open writequeue.
`OPC_ERR_CANT_INFORM_MSGM`	Informing message manager failed.
`OPC_ERR_CANT_INFORM_MSGA`	Informing message agent failed.
`OPC_ERR_SV_NOT_RUNNING`	One or more server processes are not running.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

Versions

2.00 and later

See Also

“HPOM Interfaces” on page 94

opcif_read()

```
#include opcapi.h or opcsvapi.h

int opcif_read(
                int             interface_id,     /* in */
                opcdata         data              /* in/out */
                );
```

Parameters

interface_id	Specifies which interface instance is used.
data	HPOM data structure.

Description

The function opcif_read() reads a message, message event or application response from the queue of the specified interface instance. If the interface instance is opened with OPCIF_READ_WAIT, the calling process is blocked until the information is available.

If the interface is opened with OPCIF_BUFFER and there are buffered elements when opening the interface, for example from a previous run of the application which aborted or called opcif_close_and_buffer(), then the interface is put in synchronization mode. This means that first all buffered elements are delivered one by one to the caller of this routine until no elements remain in the queue. Then the interface is switched back to normal mode. The switch from synchronization mode to normal mode is notified to the caller of this routine with the return code OPC_ERR_NO_DATA. Note that this may cause the application to consume CPU cycles since no waiting is done in opcif_read() while elements are remaining in the queue.

The API returns an error if the application receives an interrupt signal.

The data parameter specified in the call must be created with opcdata_create().

Memory for the actual message data is allocated, and if memory was assigned to data before the call to opcif_read(), it is deallocated.

Return Values

OPC_ERR_OK	OK
OPC_ERR_CANT_INIT	Initialization of queues failed.
OPC_ERR_INVALID_INTERFACE_ID	No such interface opened.

OPC_ERR_INVALID_OUTPARAM	data is NULL or is of wrong type.
OPC_ERR_WRONG_MSITYPE	Interface assigned to interface_id is of wrong type.
OPC_ERR_EINTR	Reading from pipe failed.
OPC_ERR_MSI_BUF_FULL	Number of messages exceeds specified number in max_entries while opening.
OPC_ERR_NO_DATA	Queue is empty. If the interface was opened with OPCIF_BUFFER, this return code is used to signal to the caller that synchronization mode is terminated and normal mode is resumed.
OPC_ERR_CANT_READ_MSG	Reading message, event, or response failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_MSI_BUF_FULL	Finished reading in buffered data after queue limit was reached. Switching from synchronization to normal mode. The application may continue reading from the interface as usual.

Versions

2.00 and later

See Also

“HPOM Interfaces” on page 94

“HPOM Data Structures” on page 539

opcif_register()

```
#include opcapi.h or opcsvapi.h

int opcif_register(
            int                 interface_id,     /* in */
            const opcregcond    reg_cond ,        /* in */
            long                *cond_id          /* out */
                  );
```

Parameters

`interface_id` Specifies which interface instance is used.

`reg_cond`

- *Messages*

  Defines the combination of message attributes that are checked; NULL registers for all messages.

- *Message Events*

  Defines an event mask and the restriction of message events of messages for certain operators. NULL registers for all message events.

- *Application Responses*

  Defines a certain application response specified by the application response ID. NULL is not allowed for application responses.

`cond_id` Returns an ID to reference this condition in a subsequent call to opcif_unregister(); NULL is allowed if the API user is not interested in the ID (for example, if opcif_unregister() is not called later on).

Description

The function `opcif_register()` is used by an external application to register for the following attributes. See also “OPCREGCOND” on page 588.

❑ **Message Attributes**

HPOM supports registration for message type, message group, node name, object, severity and application attributes. You can also combine attributes with a logical AND operator, and use the logical

OR operator (|) within an attribute. Multiple registrations (logical OR operator (|) of registration conditions) are also possible by using a sequence of API calls. The following attributes are supported:

- OPCREG_MSGTYPE
- OPCREG_GROUP
- OPCREG_NODENAME
- OPCREG_OBJECT
- OPCREG_SEVERITY
- OPCREG_APPLICATION

❏ **Message Event attributes**

HPOM supports the registration for an event mask and the restriction of events to certain operators. Multiple registrations for the operator restriction (logical OR operator (|) of registration conditions) can be done using a sequence of API calls. For the event mask, only one registration call is allowed. The supported attributes are:

- OPCREG_MSG_EVENT_MASK
- OPCREG_OPERATOR

❏ **Action Response attributes**

HPOM supports the registration for an application response ID, specified in the application for the call of `opcappl_start()`. For the registration of application responses, the following attributes are supported:

- OPCREG_APP_RESPONSE_ID

Return Values

OPC_ERR_OK	OK
OPC_ERR_CANT_INIT	Initialization of queues failed.
OPC_ERR_INVALID_INTERFACE_ID	No such interface opened.
OPC_ERR_CANT_INFORM_MSGM	Informing message manager failed.

Versions

2.00 and later

See Also

“HPOM Interfaces” on page 94

opcif_unregister()

```
#include opcapi.h or opcsvapi.h

int opcif_unregister(
            int             interface_id,       /* in */
            long            cond_id             /* in */
                    );
```

Parameters

`interface_id`	Specifies which interface instance is used.
`cond_id`	Specifies the registration condition to be removed; 0 unregisters for all messages, message events or application responses.

Description

To cancel prior registrations for messages, the external application calls `opcif_unregister()` with the value of `reg_cond` that was specified in the call to the registration API. As the registration mechanism is a positive filter, removing a registration condition does not mean that messages matched by this condition are filtered out after `opcif_unregister()` is called; instead, just that positive filter condition is cancelled.

By unregistering a condition for message events or application responses, information matching the unregistered condition will no longer be received.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_CANT_INIT`	Initialization of queues failed.
`OPC_ERR_INVALID_INTERFACE_ID`	No such interface opened.
`OPC_ERR_CANT_INFORM_MSGM`	Informing message manager failed.

Versions

2.00 and later

See Also

“HPOM Interfaces” on page 94

opcif_write()

```
#include opcapi.h or opcsvapi.h

int opcif_write(
            int                 interface_id,     /* in */
            const opcdata       data              /* in */
                );
```

Parameters

`interface_id`	Specifies which interface instance is used.
`data`	HPOM data structure. Must be an initialized opcdata structure of type OPCDTYPE_MESSAGE. If `data` was externally created or modified, a new message ID is assigned to it that can be queried using `opcdata_get_str()`.

Description

Use the function `opcif_write()` to write a message to the HPOM Interface. If the message was externally created or modified, the message is assigned a new message ID that can be queried after the call to `opcif_write()`.

Depending on the type of interface, the message is written into the message queue of the message manager or into the message stream within the message manager. There is no interface available that allows to write message events or application responses.

This function can only be used for interfaces of the following types:

- `OPCSVIF_EXTAGT_ACTION_RESPONSE`
- `OPCSVIF_EXTAGT_MESSAGE`
- `OPCSVIF_EXTMSGPROC_READWRITE`
- `OPCSVIF_EXTMSGPROC_WRITE`

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_CANT_INIT`	Initialization of queues failed.
`OPC_ERR_INVALID_INTERFACE_ID`	No such interface opened.

OPC_ERR_INVALID_OPCDATA_TYPE	`data` must be of type OPCDTYPE_MESSAGE or OPCDTYPE_ACTION_RESPONSE.
OPC_ERR_CANT_WRITE_MSG	Unable to write message or action response into the queue.
OPC_ERR_WRONG_MSITYPE	Interface type is invalid for this operation.

Versions

2.00 and later

See Also

“HPOM Interfaces” on page 94

“HPOM Data Structures” on page 539

opcreg_copy()

```
#include opcapi.h or opcsvapi.h

int opcreg_copy(
          const opcregcond    reg_cond,          /* in */
          opcregcond          *copy              /* out */
                )
```

Parameters

reg_cond	Address of the pointer to the registration condition.
copy	Address of the pointer to the copied condition.

Description

The function opcreg_copy() creates a complete copy of a registration condition and returns it in copy. The allocated memory has to be deallocated using opcreg_free().

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_OUTPARAM	copy is invalid.
OPC_ERR_NO_MEMORY	Allocation of memory failed.

Versions

2.00 and later

See Also

“opcreg_create()” on page 121

“opcreg_free()” on page 122

“opcreg_get_long()” on page 123

“opcreg_get_str()” on page 124

“opcreg_set_long()” on page 125

“opcreg_set_str()” on page 126

opcreg_create()

```
#include opcapi.h or opcsvapi.h

int opcreg_create(
       opcregcond           *reg_cond           /* out */
                  );
```

Parameters

`reg_cond`	Address of the pointer to the registration condition.

Description

The function `opcreg_create()` allocates and initializes an empty registration condition and returns a pointer to the created structure. The actual structure of this data area is hidden from the user. To get or set attributes, the respective routines must be called. The memory used for the area has to be deallocated by calling `opcreg_free()`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_OUTPARAM`	`reg_cond` is invalid.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.

Versions

2.00 and later

See Also

“opcreg_copy()” on page 120

“opcreg_free()” on page 122

“opcreg_get_long()” on page 123

“opcreg_get_str()” on page 124

“opcreg_set_long()” on page 125

“opcreg_set_str()” on page 126

opcreg_free()

```
#include opcapi.h or opcsvapi.h

int opcreg_free(
           opcregcond        *reg_cond       /* in/out */
               );
```

Parameters

`reg_cond`	Address of the pointer to the registration condition.

Description

The function `opcreg_free()` deallocates the memory associated with a registration condition.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_OUTPARAM`	`reg_cond` is invalid.

Versions

2.00 and later

See Also

“opcreg_copy()” on page 120

“opcreg_create()” on page 121

“opcreg_get_long()” on page 123

“opcreg_get_str()” on page 124

“opcreg_set_long()” on page 125

“opcreg_set_str()” on page 126

opcreg_get_long()

```
#include opcapi.h or opcsvapi.h

long opcreg_get_long(
         const opcregcond       reg_cond,       /* in */
         int                    field           /* in */
                    );
```

Parameters

`reg_cond`	Address of the pointer to the registration condition.
`field`	Specifies the attribute that is queried.

Description

Use the routine `opcreg_get_long()` to access the attribute values of a condition.

Return Values

Returns the integer value of the attribute; if the routine fails, -1 is returned.

Versions

2.00 and later

See Also

“opcreg_copy()” on page 120

“opcreg_create()” on page 121

“opcreg_free()” on page 122

“opcreg_get_str()” on page 124

“opcreg_set_long()” on page 125

“opcreg_set_str()” on page 126

opcreg_get_str()

```
#include opcapi.h or opcsvapi.h

char *opcreg_get_str(
    const opcregcond         reg_cond,          /* in */
    int                      field              /* in */
                    );
```

Parameters

`reg_cond`	Address of the pointer to the registration condition.
`attribute`	Specifies the attribute that is queried.

Description

Use the function `opcreg_get_str()` to access the string attribute of a registration condition.

Return Values

Returns a character pointer to the value of the defined attribute in the data area. The pointer points into the internal data area. Modification of the attribute is only allowed using opcreg_set_str(); direct access to the string is not supported, however, it is not possible to prevent the user from committing direct modifications. If not successful a NULL pointer is returned.

Versions

2.00 and later

See Also

“opcreg_copy()” on page 120

“opcreg_create()” on page 121

“opcreg_free()” on page 122

“opcreg_get_long()” on page 123

“opcreg_set_long()” on page 125

“opcreg_set_str()” on page 126

opcreg_set_long()

```
#include opcapi.h or opcsvapi.h

int opcreg_set_long(
        opcregcond      reg_cond,           /* in/out */
        int             field,              /* in */
        long            value               /* in */
                    );
```

Parameters

`reg_cond`	Address of the pointer to the registration condition.
`field`	Selects the attribute in the registration condition.
`value`	Specifies the value of the attribute in the registration condition.

Description

Use the function `opcreg_set_long()` to set the value of a numeric attribute of a registration condition.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_FIELD`	`field` is invalid.

Versions

2.00 and later

See Also

“opcreg_copy()” on page 120

“opcreg_create()” on page 121

“opcreg_free()” on page 122

“opcreg_get_long()” on page 123

“opcreg_get_str()” on page 124

“opcreg_set_str()” on page 126

opcreg_set_str()

```
#include opcapi.h or opcsvapi.h

int *opcreg_set_str(
        opcregcond        reg_cond,          /* in/out */
        int               field,             /* in */
        const char        *value             /* in */
                  );
```

Parameters

`reg_cond`	Address of the pointer to the registration condition.
`field`	Selects the attribute in the registration condition.
`value`	Specifies the value of the attribute in the registration condition.

Description

The function `opcreg_set_str()` sets the value of a string attribute of a registration condition.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_FIELD`	`field` is invalid.

Versions

2.00 and later

See Also

“opcreg_copy()” on page 120

“opcreg_create()” on page 121

“opcreg_free()” on page 122

“opcreg_get_long()” on page 123

“opcreg_get_str()” on page 124

“opcreg_set_long()” on page 125

Server Message API

HPOM provides a set of functions for the following operations:

- **Messages**

  Access and modify an active, history, or pending messages and their details.

- **Annotations**

  Get, add, or delete annotations.

- **Actions**

  Start actions by way of the API.

Data Structures

OPCDTYPE_MESSAGE

OPCDTYPE_MESSAGE_ID

OPCDTYPE_ANNOTATION

Messages and annotations are identified by their IDs.

Usage

To use the functions, include the header file `opcapi.h` or `opcsvapi.h` in your application.

The opc_connect() function allows a program to connect to the HPOM database as an HPOM user or administrator, restricting the access privileges according to the restrictions of the given operator. Depending on the configuration of the operator for whom the connection was established, some messages may not be visible, or the operator may not be allowed to add, modify, or delete annotations. In this case, the functions return `OPC_ERR_ACCESS_DENIED`.

For the connection it is not necessary that the server processes are running but when the connection is established, the application using the API will not be informed of a start of the server processes and the connected GUIs will not be informed of the changes to the messages.

If it is necessary for the application that the server processes are running, it is possible to check the connection to the display manager using opcconn_get_capability(). Only when a connection to the display manager exists are all active HPOM GUIs informed of all operations.

The functions also fail if no connection was established, the message does not exist, or the message is not accessible.

Each routine returns an error/status code and errors are logged in the files:

`/var/opt/OV/log/System.txt` (Plain text)

`/var/opt/OV/log/System.bin` (Binary)

NOTE The HTTPS agent and the management server use the same location.

If you receive an error even though the function call was successful, the running GUIs may not have been informed successfully.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see opcdata_create() or opcdata_clear()) and freeing (see opcdata_free()) the required memory.

Prerequisites

The functions are only available on the HP Operations management server.

Multithread Usage

All functions of the HPOM Connection API are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcanno_add()

```
#include opcsvapi.h

int opcanno_add (
        const opc_connection     opc_conn,     /* in */
        const opcdata            msg_id,       /* in */
        opcdata                  annotation    /* in/out */
                );
```

Parameters

opc_conn	Connection to the HPOM database.
msg_id	ID of the message the annotation will be added to; of type OPCDTYPE_MESSAGE or OPCDTYPE_MESSAGE_ID.
annotation	Data structure of type OPCDTYPE_ANNOTATION.

Description

Use the function opcanno_add() to add annotations to an existing message. The annotation is added using the operator name specified by opc_connect() as the creator of the annotation. Annotations can be added to active, pending, and history messages. The ID of the new annotation is returned in annotation.

Return Values

OPC_ERR_OK	OK, annotation was added.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL; msg_id is NULL; msg_id is not of type OPCDTYPE_MESSAGE_ID or OPCDTYPE_MESSAGE; annotation is NULL.
OPC_ERR_INVALID_ID	msg_id is malformed or database contains no message with that ID.
OPC_ERR_DATABASE_ERROR	Operator/message check failed; adding of annotation failed; information retrieval for GUI information failed.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to see this message.

OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_CANT_INFORM_UI	Failed to inform user interface.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.
OPC_ERR_OBJECT_NOT_FOUND	Annotation not found.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE” on page 561

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_ANNOTATION” on page 547

opcanno_delete()

```
#include opcsvapi.h

int opcanno_delete (
          const opc_connection     opc_conn,     /* in */
          const opcdata            msg_id,       /* in */
          opcdata                  annotation    /* in */
                   );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`msg_id`	ID of the message the annotation should be deleted from; of type OPCDTYPE_MESSAGE or OPCDTYPE_MESSAGE_ID.
`annotation`	Data structure of type OPCDTYPE_ANNOTATION. The ID of the annotation must be set in `annotation` to specify the annotation.

Description

Use the function `opcanno_delete()` to delete annotations from an existing message. Annotations can be deleted from all message types, that is from active, pending, and history messages.

Return Values

`OPC_ERR_OK`	OK, annotation was deleted.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL; `msg_id` is NULL; `msg_id` is not of type OPCDTYPE_MESSAGE_ID or OPCDTYPE_MESSAGE; `annotation` is NULL.
`OPC_ERR_INVALID_ID`	`msg_id` is malformed or database contains no message with that ID.
`OPC_ERR_DATABASE_ERROR`	Operator/message check failed; annotation could not be deleted; information retrieval for GUI information failed.
`OPC_ERR_ACCESS_DENIED`	Operator is not allowed to see this message.

OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_CANT_INFORM_UI	Failed to inform user interface.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.
OPC_ERR_OBJECT_NOT_FOUND	Annotation not found.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE” on page 561

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_ANNOTATION” on page 547

opcanno_get_list()

```
#include opcsvapi.h

int opcanno_get_list (
        const opc_connection     opc_conn,      /* in */
        const opcdata            msg_id,        /* in */
        opcdata                  annotations    /* out */
                    );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`msg_id`	ID of the message; can be OPCDTYPE_MESSAGE or OPCDTYPE_MESSAGE_ID.
`annotations`	OPCDTYPE_CONTAINER structure of type OPCDTYPE_ANNOTATION or `OPCDTYPE_EMPTY`.

Description

Use the function `opcanno_get_list()` to get a list of annotations associated with a given message. If the container already contains annotation elements, the requested annotation will be appended.

Return Values

`OPC_ERR_OK`	OK, the execution of the function was successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL; `msg_id` is NULL; `msg_id` is not of type OPCDTYPE_MESSAGE or OPCDTYPE_MESSAGE_ID.
`OPC_ERR_INVALID_OUTPARAM`	`annotations` is NULL; `annotations` is not a container of type `OPCDTYPE_EMPTY`, or of type OPCDTYPE_ANNOTATION.
`OPC_ERR_INVALID_ID`	`msg_id` is malformed or database contains no message with that ID.
`OPC_ERR_DATABASE_ERROR`	Operator/message check failed; information retrieval for GUI information failed.

OPC_ERR_ACCESS_DENIED	Operator is not allowed to retrieve information about the message.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.
OPC_WARN_NO_OBJECTS_FOUND	Message has no annotations.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE” on page 561

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_ANNOTATION” on page 547

opcanno_modify()

```
#include opcsvapi.h

int opcanno_modify (
          const opc_connection     opc_conn,     /* in */
          const opcdata            msg_id,       /* in */
          opcdata                  annotation    /* in */
                   );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`msg_id`	ID of the message the annotation should be added to; of type OPCDTYPE_MESSAGE or OPCDTYPE_MESSAGE_ID.
`annotation`	Data structure of type OPCDTYPE_ANNOTATION. The ID of the annotation must be set in `annotation` to specify the annotation.

Description

Use the function `opcanno_modify()` to modify annotations of an existing message. Annotations of active, pending, and history messages can be modified.

The annotation is identified by its ID; to modify the annotation you must first retrieve this ID using opcanno_get_list().

Return Values

`OPC_ERR_OK`	OK, annotation was added.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL; `msg_id` is NULL; `msg_id` is not of type OPCDTYPE_MESSAGE_ID or OPCDTYPE_MESSAGE; `annotation` is NULL.
`OPC_ERR_INVALID_ID`	`msg_id` is malformed or database contains no message with that ID.
`OPC_ERR_DATABASE_ERROR`	Operator/message check failed; modification of annotation failed; information retrieval for GUI information failed.

OPC_ERR_ACCESS_DENIED	Operator is not allowed to see this message.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_CANT_INFORM_UI	Failed to inform user interface.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.
OPC_ERR_OBJECT_NOT_FOUND	Annotation not found.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE” on page 561

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_ANNOTATION” on page 547

opcmsg_ack()

```
#include opcsvapi.h

int opcmsg_ack (
             opc_connection     opc_conn,        /* in */
             opcdata            message_id       /* in */
               );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	Message ID of the message to be acknowledged; of type OPCDTYPE_MESSAGE_ID.

Description

Use the function opcmsg_ack() to acknowledge messages which are currently active.

A message event is sent to all running GUIs to update them with the new information. Applications can register with the Message Event Interface to get the event OPC_MSG_EVENT_ACK.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL; message_id is NULL; message_id is not of type OPCDTYPE_MESSAGE_ID.
OPC_ERR_INVALID_ID	Database contains no message with that ID.
OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details, cannot get database information for GUI change request.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to acknowledge the message.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_CANT_INFORM_UI	Cannot inform GUI.

OPC_ERR_ALREADY_DONE	Message is already acknowledged.
OPC_ERR_MSG_OWNED_BY_ANOTHER_USER	Message is owned by another HPOM user.

Versions

3.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

opcmsg_disown()

```
#include opcsvapi.h

int opcmsg_disown (
             opc_connection     opc_conn,        /* in */
             opcdata            message_id       /* in */
                  );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	Message ID of the message to be disowned; of type OPCDTYPE_MESSAGE_ID.

Description

Use the function opcmsg_disown() to disown a message that is currently owned.

A message event is sent to all running GUIs to update them with the new information. Applications can register with the Message Event Interface to get the event OPC_MSG_EVENT_DISOWN.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL; message_id is NULL; message_id is not of type OPCDTYPE_MESSAGE_ID.
OPC_ERR_INVALID_ID	Database contains no message with that ID.
OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details, cannot get database information for GUI change request.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to disown the message.
OPC_ERR_NO_MEMORY	Cannot reserve memory for ID, cannot allocate memory for GUI information request.

OPC_ERR_CANT_INFORM_UI	Cannot inform GUI.
OPC_ERR_MSG_OWNED_BY_ANOTHER_USER	Message is owned by another HPOM user.

Versions

4.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

opcmsg_get()

```
#include opcsvapi.h

int opcmsg_get (
             opc_connection     opc_conn,     /* in */
             opcdata            message_id,   /* in */
             opcdata            message       /* out */
               );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	ID of the message information should be obtained for; of type OPCDTYPE_MESSAGE_ID.
message	HPOM data structure of type OPCDTYPE_MESSAGE.

Description

Use the function opcmsg_get() to get detailed information about a message given through a message ID. The data record in which the message information is kept is hidden within the API.

If the message found with the corresponding message ID would not normally be visible to the operator who established the connection, the API call returns OPC_ERR_ACCESS_DENIED.

The responsibility information is loaded only once when calling opc_connect. If the operator's configuration changes, this information is not reloaded automatically.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL, message_id is NULL, message_id is not of type OPCDTYPE_MESSAGE_ID, message is not of type OPCDTYPE_MESSAGE.
OPC_ERR_INVALID_ID	Database contains no message with that ID.

OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to see this message.
OPC_ERR_NO_MEMORY	Unable to allocate memory.

Versions

3.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_MESSAGE” on page 561

opcmsg_get_instructions()

```
#include opcsvapi.h

char *opcmsg_get_instructions (
                opc_connection      opc_conn,      /* in */
                opcdata             message_id     /* in */
                                );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	HPOM message ID structure; of type OPCDTYPE_MESSAGE_ID.

Description

Use the function opcmsg_get_instructions() to get instructions for a message. This function returns a pointer to the instruction string which must be freed by the caller.

Return Values

Null	If no connection was established; the operator is not allowed to see this message; no memory available; message ID is wrong; no such message found; or errors occurred in the initializing routine.

Versions

3.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

opcmsg_get_msgids()

```
#include opcsvapi.h

char *opcmsg_get_msgids (
                opc_connection      opc_conn,      /* in */
                opcdata             message_id,    /* in */
                opcdata             curr_msgids    /* out */
                                );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`message_id`	HPOM message ID structure; of type OPCDTYPE_MESSAGE_ID.
`curr_msgids`	Container OPCDTYPE_CONTAINER of type `OPCDTYPE_EMPTY` or OPCDTYPE_MESSAGE_ID.

Description

Use the function `opcmsg_get_msgids()` to get the current message identifiers to a given original message identifier.

Generally speaking, a new message ID is generated when an application copies a message from the Message Stream Interface (MSI), modifies the message, and sends the message back to the MSI. This behavior is configurable; see the *HPOM Application Integration Guide* for more information.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL, `message_id` is NULL or of wrong type, `curr_msgids` is NULL or of wrong type.
`OPC_ERR_DATABASE_ERROR`	Access to the database failed.
`OPC_ERR_NO_MEMORY`	Out of memory.
`OPC_ERR_ACCESS_DENIED`	Connected operator is not allowed to access the specified message.
`OPC_ERR_INVALID_ID`	Specified message not in database.

`OPC_ERR_INVALID_OPCDATA_TYPE`	`curr_msgids` are not of type OPCDTYPE_CONTAINER.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

opcmsg_modify()

```
#include opcsvapi.h

int opcmsg_modify (
             opc_connection     opc_conn,        /* in */
             opcdata            msg_id,          /* in */
             opcdata            mod_msg          /* in */
                 );
```

Parameters

opc_conn	Connection to the HPOM database.
msg_id	Message ID of the message to be modified; of type OPCDTYPE_MESSAGE_ID.
mod_msg	Modified message of type OPCDTYPE_MESSAGE.

Description

Use the function opcmsg_modify() to modify the message attributes OPCDATA_SEVERITY and OPCDATA_MESSAGE_TEXT of a given message. You can also use the functions of the HPOM data API to query, modify, and delete custom message attributes. Changes to other message attributes are not possible; they are ignored by HPOM.

Each successful modification is documented by an annotation that includes the name of the HPOM operator, date and time of the change, and the previous values of the attributes.

A message event is sent to all running GUIs to update them with the new information. Applications can register with the Message Event Interface to get the new event OPC_MSG_EVENT_CHANGE. See the *HPOM Application Integration Guide* for more information.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL, msg_id is NULL or of wrong type, mod_msg is NULL or of wrong type.
OPC_ERR_DATABASE_ERROR	Access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	Connected operator is not allowed to modify the specified message.

OPC_ERR_INVALID_ID	Specified message not in database.
OPC_ERR_MSG_NOT_ACTIVE	Message is not active.
OPC_ERR_APPL_REQUIRED	Definition of application required.
OPC_ERR_MSG_IS_READONLY	Message is read-only.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“opcdata_get_cma()” on page 53

“opcdata_get_cmanames()” on page 55

“opcdata_remove_cma()” on page 71

“opcdata_set_cma()” on page 73

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_MESSAGE” on page 561

opcmsg_own()

```
#include opcsvapi.h

int opcmsg_own (
             opc_connection     opc_conn,        /* in */
             opcdata            message_id       /* in */
              );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	Message ID of the message to be owned; of type OPCDTYPE_MESSAGE_ID.

Description

Use the function opcmsg_own() to own a message.

A message event is sent to all running GUIs to update them with the new information. Applications can register with the Message Event Interface to get the event OPC_MSG_EVENT_OWN.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL; message_id is NULL; message_id is not of type OPCDTYPE_MESSAGE_ID.
OPC_ERR_INVALID_ID	Database contains no message with that ID.
OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details, cannot get database information for GUI change request.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to own the message.
OPC_ERR_NO_MEMORY	Cannot reserve memory for ID, cannot allocate memory for GUI information request.
OPC_ERR_CANT_INFORM_UI	Cannot inform GUI.

OPC_ERR_MSG_OWNED_BY_ ANOTHER_USER	Message is owned by another HPOM user.

Versions

3.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_MESSAGE” on page 561

opcmsg_select()

```
#include opcsvapi.h

int opcmsg_select (
             opc_connection     opc_conn,        /* in */
             opcdata            msg_id,          /* in */
             const char*        operator_name    /* in */
                  )
```

Parameters

opc_conn	Connection to the HPOM database.
msg_id	Message ID of the message to be selected; of type OPCDTYPE_MESSAGE_ID.
operator_name	Specifies the name of the operator you wish to highlight a message for. An empty or NULL string uses the connected user as the value for this parameter.

Description

opcmsg_select() allows you to highlight a message in a particular HPOM operator's open Message Browsers. The caller of the function can specify (with a character string) the user name of the operator in whose browser the message is to be highlighted and the message ID of the message to select. The connected operator must have administrator capabilities to select a message in another operator's Message Browser(s). If no user name is specified, the select event is sent to the HPOM user that made the connection.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL msg_id is NULL or of the wrong type.
OPC_ERR_DATABASE_ERROR	Access to the database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_ACCESS_DENIED	Connected operator is not allowed to select/highlight the specified message.
OPC_ERR_INVALID_ID	Specified message not in database.

OPC_ERR_CANT_INFORM_UI	Cannot inform GUI of the select event.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_MESSAGE” on page 561

opcmsg_set_owner()

```
#include opcsvapi.h

int opcmsg_set_owner (
             opc_connection     opc_conn,        /* in */
             const opcdata      message_id,      /* in */
             const char *       operator_name    /* in */
              );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	Message ID of the message to be owned; of type OPCDTYPE_MESSAGE_ID.
operator_name	Name of the operator who will become the owner of the message.

Description

Use the function opcmsg_set_owner() to set the owner of an active message to the specified operator. The specified operator must have permission to own messages.

To use this function, it is necessary to connect to the management server as an HPOM user with administrator rights, using the function opc_connect().

A message event is sent to all running GUIs to update them with the new information. Applications can register with the Message Event Interface to get the event OPC_MSG_EVENT_OWN.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL; message_id is NULL; message_id is not of type OPCDTYPE_MESSAGE_ID.
OPC_ERR_INVALID_ID	Database contains no message with that ID, or the given ID is malformed.

OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details, cannot get database information for GUI change request.
OPC_ERR_ACCESS_DENIED	User is not allowed to own the message; user does not have administrative rights.
OPC_ERR_NO_MEMORY	Cannot reserve memory for ID, cannot allocate memory for GUI information request.
OPC_ERR_CANT_INFORM_UI	Cannot inform GUI.
OPC_ERR_MSG_OWNED_BY_ANOTHER_USER	Message is owned by another HPOM user.

Versions

8.25 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

“OPCDTYPE_MESSAGE” on page 561

opcmsg_start_auto_action()

```
#include opcsvapi.h

int opcmsg_start_auto_action (
                opc_connection      opc_conn,      /* in */
                opcdata             message_id     /* in */
                                );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	ID of the message the action should be started for; of type OPCDTYPE_MESSAGE_ID.

Description

Use the function opcmsg_start_auto_action() to restart an automatic action if one is defined for the given message. This is useful when the action was stopped or has failed.

A message event is sent to all running GUIs to update them with the new information. Applications can register with the Message Event Interface to get the event OPC_MSG_EVENT_AA_START.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL, message_id is NULL, message_id is not of type OPCDTYPE_MESSAGE_ID.
OPC_ERR_INVALID_ID	Database contains no message with that ID.
OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details, cannot get database information for GUI change request.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to start the action.
OPC_ERR_NO_ACTION_DEFINED	No automatic action defined.

OPC_ERR_ACTION_FAILED	Sending action request was not successful.
OPC_ERR_INVAL_NODE	Node not configured to execute actions.
OPC_ERR_NO_MEMORY	Cannot reserve memory for ID, cannot allocate memory for GUI information request.
OPC_ERR_CANT_INFORM_UI	Cannot inform GUI about change.
OPC_ERR_MSG_OWNED_BY_ ANOTHER_USER	Message is owned by another HPOM user.
OPC_ERR_ACTION_RUNNING	Message has a running action.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

opcmsg_start_op_action()

```
#include opcsvapi.h

int opcmsg_start_op_action (
                opc_connection      opc_conn,      /* in */
                opcdata             message_id     /* in */
                          );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	ID of the message the action should be started for; of type OPCDTYPE_MESSAGE_ID.

Description

Use the function opcmsg_start_op_action() to start an operator-initiated action if one is defined for the given message.

A message event is sent to all running GUIs to update them with the new information. Applications can register with the Message Event Interface to get the event OPC_MSG_EVENT_OA_START.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL, message_id is NULL, message_id is not of type OPCDTYPE_MESSAGE_ID.
OPC_ERR_INVALID_ID	Database contains no message with that ID.
OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details, cannot get database information for GUI change request.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to start the action.
OPC_ERR_NO_ACTION_DEFINED	No operator-initiated action defined.
OPC_ERR_ACTION_FAILED	Sending action request was not successful.

OPC_ERR_INVAL_NODE	Node not configured to execute actions.
OPC_ERR_NO_MEMORY	Cannot reserve memory for ID, cannot allocate memory for GUI information request.
OPC_ERR_CANT_INFORM_UI	Cannot inform GUI about change
OPC_ERR_MSG_OWNED_BY_ ANOTHER_USER	Message is owned by another HPOM user.
OPC_ERR_ACTION_RUNNING	Message has a running action.

Versions

4.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

opcmsg_unack()

```
#include opcsvapi.h

int opcmsg_unack (
          opc_connection     opc_conn,          /* in */
          opcdata            message_id         /* in */
                );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	Message ID of the message to be unacknowledged; of type OPCDTYPE_MESSAGE_ID.

Description

Use the function opcmsg_unack() to unacknowledge messages which are currently acknowledged (history messages).

A message event is sent to all running GUIs to update them with the new information. Applications can register with the Message Event Interface to get the event OPC_MSG_EVENT_UNACK.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL; message_id is NULL; message_id is not of type OPCDTYPE_MESSAGE_ID;
OPC_ERR_INVALID_ID	Database contains no message with that ID.
OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details, cannot get database information for GUI change request.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to unacknowledge the message.
OPC_ERR_NO_MEMORY	Cannot reserve memory for ID, cannot allocate memory for GUI information request.

OPC_ERR_CANT_INFORM_UI	Cannot inform GUI.
OPC_ERR_ALREADY_DONE	Message is already unacknowledged.

Versions

4.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

opcmsg_unbuffer()

```
#include opcsvapi.h

int opcmsg_unbuffer (
          opc_connection     opc_conn,          /* in */
          opcdata            message_id         /* in */
                );
```

Parameters

opc_conn	Connection to the HPOM database.
message_id	Message ID of the message to be unbuffered; of type OPCDTYPE_MESSAGE_ID.

Description

Use the function opcmsg_unbuffer() to unbuffer a pending message. The message will be moved from the pending messages browser to the active messages browser and the unbuffer time will be reset.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL; message_id is NULL; message_id is not of type OPCDTYPE_MESSAGE_ID.
OPC_ERR_INVALID_ID	Database contains no message with that ID.
OPC_ERR_DATABASE_ERROR	Cannot get operator/message capabilities, cannot get message details, cannot get database information for GUI change request.
OPC_ERR_ACCESS_DENIED	Operator is not allowed to unbuffer the message.
OPC_ERR_NO_MEMORY	Cannot reserve memory for ID, cannot allocate memory for GUI information request.
OPC_ERR_CANT_INFORM_UI	Cannot inform GUI.
OPC_ERR_ALREADY_DONE	Message is already unbuffered.

OPC_ERR_MSG_OWNED_BY_ANOTHER_USER	Message is owned by another user.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“OPCDTYPE_MESSAGE_ID” on page 572

Agent Message API

HPOM provides a set of APIs to handle messages on managed nodes. These functions enable you, for example, to send messages and acknowledge them at a later time. See “Agent Monitor API” on page 169 for functions to send monitor values.

Data Structures

OPCDTYPE_MESSAGE_ID

OPCDTYPE_MESSAGE

Messages are identified by their IDs.

Usage

The managed node processes must be running. To use the functions, include the header file `opcapi.h` or `opcsvapi.h` in your application.

Prerequisites

The functions can only be called by the user root or the user/group to which the HP Operations agent belongs (in the case of a non-root agent).

Each opcdata structure must be allocated using opcdata_create() before it can be used in any of these functions. After the execution of your program, each opcdata structure must be freed using opcdata_free().

Multithread Usage

All functions of the Agent Message API are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

Agent Configuration

Operations on messages out of managed nodes require to send these message operations to the manager. Unfortunately it is not possible to deliver the responsible manager of a message from the message ID. Additionally the configuration could be changed since the message was sent so that it is necessary to send the message operation to all managers. This can produce a lot of network load.

To prevent this, the message agent holds information about the manager to which the messages were sent. After a defined time the information will be deleted to save memory, disk space and processing time. This time is configurable:

- **HTTPS-based managed nodes**

```
ovconfchg -ns eaagt -set OPC_STORE_TIME_FOR_MGR_INFO \
<time_in_hours>
```

The specified value is the time in hours with a default setting of one hour if this parameter is not changed.

The storage of the manager information must be enabled for each message to be sent by setting the message parameter `OPCDATA_DATA_INFO` to `OPC_REMARK_FOR_ACK`.

```
...
opcdata_set_long (message, OPCDATA_DATA_INFO,
OPC_REMARK_FOR_ACK);
...
```

opcagtmsg_ack()

```
#include opcapi.h or opcsvapi.h

int opcagtmsg_ack (
                opcdata     message_id     /* in */
                  );
```

Parameters

`message_id` Message ID of type OPCDTYPE_MESSAGE_ID.

Description

Use the function `opcagtmsg_ack()` to acknowledge a message out from a managed node. Therefore a message operation will be sent to the message agent and forwarded to the message interceptor.

If the message attribute `OPCDATA_DATA_INFO` of a previously sent message was set to `OPC_REMARK_FOR_ACK`, the message agent holds the information about the responsible manager in its memory. If this attribute was not set, the message operation will be sent to all managers.

The API program must run as the user root or the agent user/group to which the HP Operations agent belongs (in the case of a non-root agent). If not, your program must set the user ID (setuid).

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`message_id` is NULL.
`OPC_ERR_INVALID_OPCDATA_TYPE`	`message_id` is not of type OPCDTYPE_MESSAGE_ID.
`OPC_ERR_INCOMPLETE_PARAM`	Message ID is not set.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

Versions

4.00 and later

See Also

“opcagtmsg_send()” on page 165

“opcmsg()” on page 167

“OPCDTYPE_MESSAGE_ID” on page 572

opcagtmsg_send()

```
#include opcapi.h or opcsvapi.h

int opcagtmsg_send (
                opcdata     message     /* in/out */
                  );
```

Parameters

`message`	Message of type OPCDTYPE_MESSAGE. The message object must be filled correctly with opcdata_set_long() and opcdata_set_str().

Description

Use the function `opcagtmsg_send()` to send a message that is created on the managed node to the responsible manager. The message ID can be retrieved from the message object using opcdata_get_str() immediately after the send call was executed:

```
opcdata_get_str(message, OPCDATA_MSGID)
```

Only the following message attributes are used in `opcagtmsg_send()`:

- `OPCDATA_SEVERITY` (Severity)
- `OPCDATA_APPLICATION` (Application)
- `OPCDATA_GROUP` (Message Group)
- `OPCDATA_OBJECT` (Object)
- `OPCDATA_MSGTEXT` (Message Text)
- `OPCDATA_NODENAME` (Node)
- `OPCDATA_OPTION_VAR` (Option Strings)

It is also possible to set other attributes but these will be ignored by the function. To forward the message object, it must contain at least the following attributes:

- `OPCDATA_APPLICATION` (Application)
- `OPCDATA_OBJECT` (Object)
- `OPCDATA_MSGTEXT` (Message Text)

The API program must run as the user root or the agent user/group to which the HP Operations agent belongs (in the case of a non-root agent). If not, your program must set the user ID (setuid).

If you want to save the information about the responsible manager, remark the message to be acknowledged later. To do this, set `OPCDATA_DATA_INFO` to `OPC_REMARK_FOR_ACK`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_APPL_REQUIRED`	Attribute `OPCDATA_APPLICATION` not set.
`OPC_ERR_OBJ_REQUIRED`	Attribute `OPCDATA_OBJECT` not set.
`OPC_ERR_TEXT_REQUIRED`	Attribute `OPCDATA_MSGTEXT` not set.
`OPC_ERR_INVAL_SEVERITY`	Set severity invalid.
`OPC_ERR_MISC_NOT_ALLOWED`	Message group `Misc` not allowed.
`OPC_ERR_INVALID_INPARAM`	`message` is NULL `message` is not of type OPCDTYPE_MESSAGE.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

Versions

4.00 and later

See Also

“opcagtmsg_ack()” on page 164

“opcmsg()” on page 167

“OPCDTYPE_MESSAGE” on page 561

opcmsg()

```
#include opcapi.h or opcsvapi.h

int opcmsg (
             const int      severity,      /* in */
             const char *   application,   /* in */
             const char *   object,        /* in */
             const char *   msg_text,      /* in */
             const char *   msg_group,     /* in */
             const char *   nodename,      /* in */
            );
```

Parameters

`severity` Severity level of the new message. Possible values are:

- `OPC_SEV_NORMAL`
- `OPC_SEV_WARNING`
- `OPC_SEV_MINOR`
- `OPC_SEV_MAJOR`
- `OPC_SEV_CRITICAL`

`application` Name of application (or script/program) which has been affected by, or has detected the event or problem. A maximum length of 32 bytes applies.

`object` Object which has been affected by, or has detected the event or problem. A maximum length of 32 bytes applies.

`msg_text` Descriptive text explaining the event or problem in more detail.

`msg_group` Default message group to which the message will belong. By default, no message group is assigned. A maximum length of 32 bytes applies.

`nodename` System from which the event problem has arisen. If passing `OPC_LOCAL_NODE`, the node name of the current system is applied.

Description

Use the function `opcmsg()` to submit a message to HPOM. The message is interpreted by the HPOM message interceptor running on the local managed node. Depending on the configuration of the message interceptor, the message is either discarded, logged locally, forwarded to the management server, or forwarded to the management server with local logging.

The message interceptor must be configured and running on the managed node, otherwise the function `opcmsg()` will fail.

This function does not return the message ID so that it is not possible to acknowledge the message later, on the managed node.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_APPL_REQUIRED`	Attribute `OPCDATA_APPLICATION` not set.
`OPC_ERR_OBJ_REQUIRED`	Attribute `OPCDATA_OBJECT` not set.
`OPC_ERR_TEXT_REQUIRED`	Attribute `OPCDATA_MSGTEXT` not set.
`OPC_ERR_INVAL_SEVERITY`	Set severity invalid.
`OPC_ERR_MISC_NOT_ALLOWED`	Message group `Misc` not allowed.
`OPC_ERR_NO_MEMORY`	Out of memory.

Versions

2.00 and later

See Also

“opcagtmsg_ack()” on page 164

“opcagtmsg_send()” on page 165

Agent Monitor API

HPOM provides a set of functions to send monitor values.

Data Structures

OPCDTYPE_MONITOR_MESSAGE

Usage

To use these functions, the managed node processes must be running. To use the functions, include the header file `opcapi.h` or `opcsvapi.h` in your application.

Prerequisites

The functions can only be called by the the user root or the user/group to which the HP Operations agent belongs (in the case of a non-root agent).

Each opcdata structure must be allocated using opcdata_create() before it can be used in any of these functions.

Multithread Usage

All functions of the HPOM Connection API are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcagtmon_send()

```
#include opcapi.h or opcsvapi.h

int opcagtmon_send (
              opcdata     mon_msg     /* in */
                  );
```

Parameters

`mon_msg` — Monitor message/value of type OPCDTYPE_MONITOR_MESSAGE. This structure must be created using opcdata_create() and must be filled with at least the monitored object name `OPCDATA_MON_VAR` and the monitor value `OPCDATA_MON_VALUE`. Additionally, the message object `OPCDATA_OBJECT` and the optional variables `OPCDATA_OPTION_VAR` can be set.

Description

The function `opcagtmon_send()` forwards the current value of the monitored object to the HPOM monitor agent running on the local managed node.

The monitor agent checks this value against the configured threshold. If the threshold is exceeded, the event is locally logged, suppressed, or forwarded to the message agent running on the local system, depending upon the threshold monitor configuration.

The message agent forwards the message to the HP Operations management server where it can be viewed in the HPOM message browser window by the responsible HPOM operators.

If a local automatic action has been set up to run when a threshold is exceeded, this action is performed immediately by the local HPOM action agent.

The monitor agent must be configured and running on the managed node, otherwise the function `opcagtmon_send()` will fail.

Only the message attributes Monitor Name, Monitor Value, Object and Option String are used in `opcagtmon_send()`.

The API program must be run as the user root or the agent user/group to which the HP Operations agent belongs (in the case of a non-root agent).. If not, your program must set the user ID (setuid).

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	mon_msg is NULL mon_msg is not of type OPCDTYPE_MONITOR_MESSAGE.
OPC_ERR_OBJNAME_REQUIRED	Attribute OPCDATA_MON_VAR not set.
OPC_ERR_NO_AGENT	Agent is not running.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_WRONG_OPTION_VARS	Attribute OPCDATA_OPTION_VAR not set correctly.

Versions

4.00 and later

See Also

“opcmon()” on page 172

“OPCDTYPE_MONITOR_MESSAGE” on page 573

opcmon()

```
#include opcapi.h or opcsvapi.h

int opcmon (
          const char     *objname,     /* in */
          const double   monval        /* in */
           );
```

Parameters

`objname`	Name of the monitored object as configured in the `Object Pattern` field of threshold monitor conditions.
`monval`	Current value of the monitored object.

Description

The function `opcmon()` forwards the current value of the monitored object to the HPOM monitor agent running on the local managed node. The monitor agent checks this value against the configured threshold. If the threshold is exceeded, the event is locally logged, suppressed, or forwarded to the message agent running on the local system, depending upon the threshold monitor configuration.

The message agent forwards the message to the HP Operations management server where it can be viewed in the HPOM message browser window by the responsible HPOM operators.

If a local automatic action has been set up to run when a threshold is exceeded, this action is performed immediately by the local HPOM action agent.

The monitor agent must be configured and running on the managed node, otherwise the function opcmon will fail.

The API program must run as the user root or the agent user/group to which the HP Operations agent belongs (in the case of a non-root agent). If not, your program must set the user ID (setuid).

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_OBJNAME_REQUIRED`	`objname` is NULL.
`OPC_ERR_NO_AGENT`	Agent is not running.
`OPC_ERR_NO_MEMORY`	Out of memory.

Versions

2.00 and later

See Also

“opcagtmon_send()” on page 170

3 Functions of the HPOM Configuration APIs

In This Chapter

The **HPOM Configuration API** provides functions which allow you to access the HPOM configuration, for example, to get a list of the managed nodes belonging to a specific operator.

- ❏ Connection API
- ❏ Application Configuration API
- ❏ Application Group Configuration API
- ❏ Message Group Configuration API
- ❏ Message Regroup Condition Configuration API
- ❏ Node Configuration API
- ❏ Node Hierarchy Configuration API
- ❏ Policy Configuration API
- ❏ Category Configuration API
- ❏ User Profile Configuration API
- ❏ User Configuration API
- ❏ Distribution API
- ❏ Server Synchronization API

NOTE The file `opcsvapi.h` contains predefined values for the function parameters, the function prototypes, and defines the error codes. The file is located in: `/opt/OV/include/`.

NOTE The HPOM synchronization functionality provides that the changes performed by the operations that change the database, like add, delete, modify, assign, deassign, are visible in the Java GUI immediately.

Configuration API Usage

The functions opc*_modify() always modify *all* attributes of the HPOM object. Therefore, when using a modify function, you must always set all attributes, including the attributes that didn't change.

As there are read-only or write-only attributes which you cannot set during a modify operation, you must first fill the opcdata structure with an opc*_get() or opc*_get_list() operation so that the attributes you didn't modify aren't overwritten.

The out parameter returned by the opc*_get() operation can then be set to a new value and returned as the in parameter of the opc*_modify() function.

Only the out parameter of the opc*_get_list() operation can be used to specify the object to be modified; the other parameters *cannot* be used to specify object attributes for a modify operation.

All configuration APIs are blocked while `opccfgupld` is running. The exceptions are the opc*_get() API calls and the API calls that are called from outside of the transaction block (not within `opctransaction_start` and `opctransaction_stop`).

Blocking mode is controlled via the `OPC_CFGUPLD_BLOCK_RETRY` configuration variable, which can have one of the following values:

- 1: exit immediately with error if `opccfgupld` is running (default).
- N: repeat N times before exiting with error.
- -1: repeat indefinitely, wait until `opccfgupld` finishes, then perform API operation.

Example of an Object Modification

The following example modifies the label of an HPOM user: the function `opcuser_get()` first gets the name of the user and returns it as the parameter `user_conf`; `opcdata_set_str()` sets the attribute to a new value; `opcuser_modify()` then modifies the parameter `user_conf`.

```
int myUserModify(opc_connection opc_conn, char *userName,
    char *newLabel)
{
   int                 rc;
   opcdata             user      = NULL;
   opcdata             user_conf = NULL;


   /* init opcdata structures */
   rc = opcdata_create(OPCDTYPE_USER_CONFIG, &user);
   rc = opcdata_create(OPCDTYPE_USER_CONFIG, &user_conf);

   /* get attributes of user with name 'userName' */

   /* first set name attribute */
   rc = opcdata_set_str(user, OPCDATA_NAME, userName);
   rc = opctransaction_start (opc_conn);

   /* next get the user (by name) */
   rc = opcuser_get(opc_conn, user, user_conf);

   /* change the label (or anything else you want ...) */
   rc = opcdata_set_str(user_conf, OPCDATA_LABEL, newLabel);
   rc = opctransaction_commit (opc_conn);

   /* modify this user */
   rc = opcuser_modify(opc_conn, user, user_conf);

   return 0;

}
```

Connection API

The Connection API provides a set of functions to handle connections to the HP Operations management server.

A connection to the management server must be established before any of the HPOM Configuration API functions can be used. Without a connection, the Configuration API functions will fail.

Memory for the configuration data is allocated on the heap.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Data Structures

`opc_connection`

Usage

The Connection API can be issued by any user.

The HPOM database must be up and running on the management server, otherwise the function `opc_connect()` will fail. It will also fail if the operator does not exist, or the given password is wrong.

To connect to HPOM, it is not necessary that all server processes are running. However, if the display manager is not running during the connection process, the connecting application will not be able to inform the connected GUIs about configuration or message changes. To check the status of the display manager during the connection process, use the function opcconn_get_capability() to check the attribute `OPCDATA_DM_CONNECTED` in `opc_conn`.

The connection returned in the `opc_connection` data structure contains all necessary information about the connected user. This includes also information about started transactions which will be committed latest when opc_disconnect() is called.

Prerequisites

The connection API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Connection API are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

Because of the internal structure of the HPOM server library, each connection creates threads, also when the connecting application in not threaded.

When opc_connect() is used in multithreaded applications, each thread must be connected separately to make sure that the server synchronization (transactions) can be handled correctly.

Another problem may arise when the application is connected and then running in threads. If opc_disconnect() is called in one of these threads, *all* threads are disconnected and each API call will fail within these threads.

opc_connect()

```
   #include opcsvapi.h

int opc_connect (
             const char         *operator_name,      /* in */
             const char         *operator_passwd,    /* in */
             opc_connection     *opc_conn            /* out */
                 );
```

Parameters

operator_name	Name of the operator.
operator_passwd	Password of the operator.

These parameters define the operator who is permitted to establish a connection. If the given password is NULL, the password check for the specified LTU user is suppressed. This is only successful if the UID of the calling application is not NULL. The operator_name can also be given as opc_adm. If so, no restrictions according to the responsibility matrix are placed upon the handling of messages.

opc_conn	This returned value has to be used in subsequent calls to this API to refer to this connection.

Description

Use the function opc_connect() to connect to the HPOM database and the display manager, and to return a connection descriptor which must be specified for further calls. The structure of the connection descriptor is hidden from the user. The memory used for this area must be deallocated by calling opc_disconnect().

This routine will itself allocate memory which has to be freed using opc_disconnect(). If a return value different from OPC_ERR_OK is returned then the opc_conn information is set to NULL and all memory is freed up. No additional call to opc_disconnect() is necessary.

Connecting via the API will not inhibit the same operator from logging into the HPOM GUI. Several programs can connect to HPOM using the same operator.

Return Values

OPC_ERR_INVALID_INPARAM	operator_name is NULL.
OPC_ERR_INVALID_OUTPARAM	opc_conn is NULL.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CANT_INIT	Cannot initialize opc_conn, cannot initialize server program.
OPC_ERR_CANT_CONNECT_DB	Connection to database failed.
OPC_ERR_NO_LOGIN	Login failed.
OPC_ERR_DATABASE_ERROR	Cannot access database information.
OPC_ERR_NO_OPERATOR_DEF	Cannot retrieve operator definition.
OPC_ERR_CANT_CONNECT_DM	Cannot connect to display manager.
OPC_ERR_OK	OK

Versions

3.00 and later

See Also

“opc_disconnect()” on page 183

“opcconn_get_capability()” on page 184

“opcconn_set_capability()” on page 187

opc_disconnect()

```
#include opcsvapi.h

int opc_disconnect (
           opc_connection    *opc_conn       /* in/out */
                   );
```

Parameters

opc_conn	Internal HPOM connection information. Must be created by opc_connect().

Description

Use the function opc_disconnect() to disconnect from the HPOM database, commit all open transactions, and to free the memory used for the connection information. This must be called if the connection created with opc_connect() is no longer needed.

Return Values

OPC_ERR_OK	Disconnected successfully.
OPC_ERR_INVALID_OUTPARAM	opc_conn is NULL.
OPC_ERR_CANT_DISCONNECT	Cannot disconnect from database.

Versions

3.00 and later

See Also

“opc_connect()” on page 181

“opcconn_get_capability()” on page 184

“opcconn_set_capability()” on page 187

opcconn_get_capability()

```
   #include opcsvapi.h

int opcconn_get_capability (
             opc_connection        opc_conn,        /* in */
             int                   attrib,          /* in */
             void*                 value            /* out */
                 );
```

Parameters

`opc_conn`: Internal HPOM connection information. Must be created by opc_connect().

`attrib` Attribute that is queried. See below for a list of available attributes and their return values.

`value` Returned value for the requested attribute.

Description

Allows access to the following capabilities of the connected HPOM operator:

`OPCDATA_ACKNOWLEDGE` (int)

`TRUE` or `FALSE`. Returns whether the operator is able to acknowledge messages.

`OPCDATA_AUDIT_LEVEL` (int)

Internal use only.

`OPCDATA_CHANGE_MSG` (int)

`TRUE` or `FALSE`. Returns whether the operator is able to change message attributes.

`OPCDATA_CSI_STRING` (int)

Configuration Stream Interface (CSI) client name. Returns the name of the client that opened a CSI with `opcif_open()` call. Used to prevent this client to get back it's own configuration changes.

`OPCDATA_DM_CONNECTED` (int)

TRUE or FALSE. Contains the connection status to the HPOM display manager. If the display manager was running during the connection process, TRUE will be returned.

OPCDATA_ID (char *)

The HPOM user's UUID.

OPCDATA_IMMEDIATE_SYNC (int)

TRUE or FALSE. Returns whether the database is synchronized immediately after each API call or not.

OPCDATA_NOOWN_NOANNO_MSG_MODIFY (int)

TRUE or FALSE. Returns whether the operator is unable to own messages and add annotations, but is able to change message attributes.

OPCDATA_OWN (int)

TRUE or FALSE. Returns whether the operator is able to own and disown messages.

OPCDATA_PASSWORD (char *)

Hexadecimal representation of the encrypted password.

OPCDATA_NAME (char *)

The HPOM user's user name.

OPCDATA_PERFORM_ACTION (int)

TRUE or FALSE. Returns whether the operator is able to perform operator-initiated actions.

OPCDATA_REAL_NAME (char *)

The HPOM user's real name.

OPCDATA_USER_ROLE

Role of the connected user. Possible values are:

- OPC_ROLE_UNKNOWN
- OPC_ROLE_OPERATOR
- OPC_ROLE_ADMINISTRATOR
- OPC_ROLE_ADMIN_OPER

- OPC_ROLE_TEMPLATE_ADMIN
- OPC_ROLE_PROFILE

Return Values

OPC_ERR_INVALID_INPARAM	opc_conn or attrib is NULL.
OPC_ERR_INVALID_OUTPARAM	value is NULL.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CANT_INIT	Cannot initialize opc_conn, cannot initialize server program.
OPC_ERR_CANT_CONNECT_DB	Connection to database failed.
OPC_ERR_NO_LOGIN	Login failed.
OPC_ERR_DATABASE_ERROR	Cannot access database information.
OPC_ERR_NO_OPERATOR_DEF	Cannot retrieve operator definition.
OPC_ERR_CANT_CONNECT_DM	Cannot connect to display manager.
OPC_ERR_OK	OK

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“opcconn_set_capability()” on page 187

opcconn_set_capability()

```
   #include opcsvapi.h

int opcconn_set_capability (
             opc_connection        opc_conn,        /* in */
             int                   attrib,          /* in */
             void*                 value            /* in */
                 );
```

Parameters

opc_conn	Internal HPOM connection information. Must be created by opc_connect().
attrib	Attribute that is set. See below for a list of available attributes.
value	Value for the attribute to be set.

Description

Allows to set the following capabilities of the connection:

OPCDATA_CSI_STRING (int)

Configuration Stream Interface (CSI) client name. Returns the name of the client that opened a CSI with opcif_open() call. Used to prevent this client to get back it's own configuration changes.

OPCDATA_IMMEDIATE_SYNC (int)

TRUE or FALSE. Defines whether the database is synchronized immediately after each API call or not.

OPCDATA_NOOWN_NOANNO_MSG_MODIFY (int)

TRUE or FALSE. Defines whether the operator is unable to own messages and add annotations, but is able to change message attributes.

Return Values

OPC_ERR_INVALID_INPARAM	opc_conn, attrib, value is NULL.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CANT_INIT	Cannot initialize opc_conn, cannot initialize server program.

OPC_ERR_CANT_CONNECT_DB	Connection to database failed.
OPC_ERR_NO_LOGIN	Login failed.
OPC_ERR_DATABASE_ERROR	Cannot access database information.
OPC_ERR_NO_OPERATOR_DEF	Cannot retrieve operator definition.
OPC_ERR_CANT_CONNECT_DM	Cannot connect to display manager.
OPC_ERR_OK	OK

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

“opcconn_get_capability()” on page 184

Application Configuration API

The application API provides a set of functions to configure HPOM applications.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see “opcdata_create()” on page 49 or “opcdata_clear()” on page 46) and freeing (see “opcdata_free()” on page 51) the needed memory.

Data Structures

`OPCDTYPE_APPL_CONFIG`

An application is specified either by name or by the UUID. If the UUID is given, a specified name will be ignored.

Usage

To use these functions, it is necessary to connect to the HP Operations management server as an HPOM user with administrator rights, using the function opc_connect() (see page 181). Only the function opcappl_start() can also be called by any other user.

Prerequisites

The Application Configuration API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcappl_add()

```
#include opcsvapi.h

int opcappl_add (
      opc_connection opc_conn,  /* in/out */
      opcdata        parentgrp, /*     in */
      opcdata        appl       /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`parentgrp`	Application group configuration of type OPCDTYPE_APPL_GROUP.
`appl`	Application configuration of type OPCDTYPE_APPL_CONFIG.

Description

Use the function `opcappl_add()` to add a given application to the HPOM database. To add the application, the name of the application must be specified. To ensure that all necessary fields are filled correctly, the application configuration is verified before the application is added. If an application with this name already exists, `OPC_ERR_OBJECT_ALREADY_EXISTS` is returned and the application is not created.

If the function completes successfully, the UUID of the application is returned in `appl`.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Use “opcapplgrp_assign_appls()” on page 207 to assign your new application to an application group.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	appl is NULL or invalid type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Application name already exists.
OPC_ERR_INVALID_NODE	Node not valid.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

“opcapplgrp_assign_appls()” on page 207

“opcdata_create()” on page 49

opcappl_delete()

```
#include opcsvapi.h

int opcappl_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        appl      /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
appl	Application configuration of type OPCDTYPE_APPL_CONFIG

Description

Deletes the specified application.

appl must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or appl is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPL_NOT_FOUND	Application not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“opc_connect()” on page 181

opcappl_get()

```
#include opcsvapi.h

int opcappl_get (
      opc_connection opc_conn,  /* in/out */
      const opcdata  appl,      /*     in */
      opcdata        appl_conf  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`appl`	Application configuration of type OPCDTYPE_APPL_CONFIG.
`appl_conf`	Application configuration of type OPCDTYPE_APPL_CONFIG.

Description

Gets the full configuration of the application specified in `appl`.

The application must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

`appl` and `appl_conf` must be data structures of the type OPCDTYPE_APPL_CONFIG.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `appl` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`appl_conf` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.

OPC_ERR_APPL_NOT_FOUND	Application not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“opc_connect()” on page 181

opcappl_get_list()

```
#include opcsvapi.h

int opcappl_get_list (
      opc_connection opc_conn,  /* in/out */
      opcdata        appl_list  /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
appl_list	Container of type OPCDTYPE_CONTAINER or OPCDTYPE_APPL_CONFIG. The returned elements are of type OPCDTYPE_APPL_CONFIG.

Description

Returns the configuration of all applications in the application bank.

The parameter appl_list must be an OPCDTYPE_CONTAINER of the type OPCDTYPE_EMPTY or OPCDTYPE_APPL_CONFIG.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	appl_list is NULL or invalid type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_OBJECT_NOT_FOUND	Application bank not found.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_CONTAINER” on page 542

“opc_connect()” on page 181

opcappl_modify()

```
#include opcsvapi.h

int opcappl_modify (
      opc_connection opc_conn,  /* in/out */
      const opcdata  appl,      /*     in */
      opcdata        appl_conf  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`appl`	Application configuration of type OPCDTYPE_APPL_CONFIG.
`appl_conf`	Application configuration of type OPCDTYPE_APPL_CONFIG.

Description

Modifies the specified application. The application must be specified by either the UUID or the name. If the UUID is given, the name will be ignored. The full configuration data will be set.

`appl_conf` must contain the full new configuration.

The application configuration is checked before modification. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

The name of the application must be specified. If an application with this name already exists, `OPC_ERR_OBJECT_ALREADY_EXISTS` is returned and the application will not be created.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `appl` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`appl_conf` is NULL or invalid or attempt to modify `OPCDATA_APP_TYPE`.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPL_NOT_FOUND	Application not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_OBJECT_ALREADY_EXISTS	Name of appl_conf already exists.
OPC_ERR_INVALID_NODE	Node not valid.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“opc_connect()” on page 181

opcappl_start()

```
#include opcsvapi.h

int opcappl_start (
      const opc_connection opc_conn,    /*     in */
      const opcdata        application, /*     in */
      const opcdata        nodes,       /*     in */
      const char *         app_exec_id  /*     in */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`application`	OPCDTYPE_APPL_CONFIG structure defines the application to start.
`nodes`	OPCDTYPE_CONTAINER containing all `nodes` of the type OPCDTYPE_NODE on which the application will be started.
`app_exec_id`	Contains the UUID to identify the application for the Application Response Interface.

Description

Sends application execution requests from the application specified in `application` to the nodes listed in `nodes`. When the request is sent successfully, the function returns `OPC_ERR_OK`.

The application must be specified by either the UUID or the name. If the UUID is given, the name will be ignored. You must also specify the user name and password of the user under which permissions the application will be started.

To call this function, it is not necessary to be logged in as administrator. Other users can only start such applications which are configured for them. Otherwise `OPC_ERR_ACCESS_DENIED` will be returned.

The application execution UUID `app_exec_id` is a regular string and can contain up to 36 characters. This UUID is needed to receive the application response from the HPOM Application Response Interface (see also opcif_api(3)). The function is not able to return the application response directly.

The HP Operations agent software must be installed and running on the managed node. Otherwise the application cannot be executed.

See also the following example program for more information about this function:

`/opt/OV/OpC/examples/progs/itoapp_start.c`

Return Values

`OPC_ERR_OK`	Execution request successfully sent.
`OPC_ERR_INVALID_INPARAM`	`opc_conn`, `application` or `nodes` is NULL `appl_exec_id` is NULL or has zero length.
`OPC_ERR_INVALID_OPCDATA_TYPE`	`application` or `nodes` has the wrong type.
`OPC_ERR_INVALID_NODE`	Node is not configured for the connected user.
`OPC_ERR_DATABASE_ERROR`	Database access failed.
`OPC_ERR_ACCESS_DENIED`	Application is not assigned to the connected user.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_INVALID_APPLICATION`	Application not found in the database.
`OPC_ERR_INVALID_APP_PARAM`	Application parameter contains a semicolon (;).

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_NODE” on page 574

“opc_connect()” on page 181

Application Group Configuration API

The application group API provides a set of functions to configure HPOM application groups.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see opcdata_create(3) or opcdata_clear(3)) and freeing (see opcdata_free(3)) the needed memory.

Data Structures

`OPCDTYPE_APPL_CONF`

An application group is specified either by name or by the UUID. If the UUID is given, a specified name will be ignored.

Usage

To use these functions, it is necessary to connect to the HP Operations management server as an HPOM user with administrator rights, using the function opc_connect() (see page 181).

Prerequisites

The Application Group Configuration API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcapplgrp_add()

```
#include opcsvapi.h

int opcapplgrp_add (
      opc_connection opc_conn,  /* in/out */
      opcdata        parentgrp, /*     in */
      opcdata        applgrp    /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
parentgrp	Application group configuration of type OPCDTYPE_APPL_GROUP.
applgrp	Application group configuration of type OPCDTYPE_APPL_GROUP.

Description

Adds the specified application group to the defined parent group. The parent group must be specified by UUID or name. The application group configuration is checked before creation. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition. The name of the application group must be specified.

The UUID of the created object will be returned in the opcdata structure if successful.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	applgrp is NULL or invalid type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.

`OPC_ERR_OBJECT_ALREADY_EXISTS`	Application group name already exists.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

“opcdata_create()” on page 49

opcapplgrp_assign_applgrps()

```
#include opcsvapi.h

int opcapplgrp_assign_applgrps (
      opc_connection opc_conn,     /* in/out */
      const opcdata  applgrp,      /*     in */
      opcdata        applgrp_list  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
applgrp	Application group configuration of type OPCDTYPE_APPL_GROUP.
applgrp_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Assigns application groups to a specified application group. Only a link to the given application group will be created.

The application group must be specified by either the UUID or the name. If the UUID is given, the name will be ignored. The same application group can have more than one valid absolute name if the application or a parent application group is assigned also to other application groups.

If an application group could not be assigned to the parent application group, OPC_ERR_NOT_COMPLETELY_DONE is returned and the specific error code is in the OPCDATA_STATUS field of the application group opcdata structure.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or applgrp is NULL.
OPC_ERR_INVALID_OUTPARAM	applgrp_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPLGROUP_NOT_FOUND	Application group not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all application groups could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Application group already assigned in the application group.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

opcapplgrp_assign_appls()

```
#include opcsvapi.h

int opcapplgrp_assign_appls (
      opc_connection opc_conn,  /* in/out */
      const opcdata  applgrp,   /*     in */
      opcdata        appl_list  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
applgrp	Application group configuration of type OPCDTYPE_APPL_GROUP.
appl_list	Container of type OPCDTYPE_APPL_CONFIG.

Description

Assigns applications to a specified application group. Only a link to the given application will be created. The application group must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The same application can have more than one parent application group.

If an application could not be assigned to the application group OPC_ERR_NOT_COMPLETELY_DONE is returned and the specific error code is in the OPCDATA_STATUS field of the application opcdata structure.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or applgrp is NULL.
OPC_ERR_INVALID_OUTPARAM	appl_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.

OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPLGROUP_NOT_FOUND	Application group not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all applications could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Application already assigned in the applgrp.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

opcapplgrp_deassign_applgrps()

```
#include opcsvapi.h

int opcapplgrp_deassign_applgrps (
      opc_connection opc_conn,     /* in/out */
      const opcdata  applgrp,      /*     in */
      opcdata        applgrp_list  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
applgrp	Application group configuration of type OPCDTYPE_APPL_GROUP.
applgrp_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Deassigns application groups from a specified application group. Only a link to the given application group will be removed. The application group must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If an application group could not be deassigned from the parent application group, OPC_ERR_NOT_COMPLETELY_DONE is returned and the specific error code is in the OPCDATA_STATUS field of the application group opcdata structure.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or applgrp is NULL.
OPC_ERR_INVALID_OUTPARAM	applgrp_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.

OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPLGROUP_NOT_FOUND	Application group not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all application groups could be assigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Application group is not assigned in the application group.
OPC_ERR_LAST_REFERENCE	Deassigning this application group would remove the last reference of this application group. This is not allowed.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

opcapplgrp_deassign_appls()

```
#include opcsvapi.h

int opcapplgrp_deassign_appls (
      opc_connection opc_conn,  /* in/out */
      const opcdata  applgrp,   /*     in */
      opcdata        appl_list  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
applgrp	Application group configuration of type OPCDTYPE_APPL_GROUP.
appl_list	Container of type OPCDTYPE_APPL_CONFIG.

Description

Deassigns applications from a specified application group. Only a link to the given application will be removed. The application group must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If an application could not be deassigned from the application group, OPC_ERR_NOT_COMPLETELY_DONE is returned and the specific error code is in the OPCDATA_STATUS field of the application opcdata structure.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or applgrp is NULL.
OPC_ERR_INVALID_OUTPARAM	appl_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.

OPC_ERR_APPLGROUP_NOT_FOUND	Application group not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all applications could be assigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Application is not assigned in the application group.
OPC_ERR_LAST_REFERENCE	Deassigning this application would remove the last reference of this application. This is not allowed.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

opcapplgrp_delete()

```
#include opcsvapi.h

int opcapplgrp_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        applgrp   /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
applgrp	Application group configuration of type OPCDTYPE_APPL_GROUP.

Description

Deletes the specified application group. The application group must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or applgrp is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPLGROUP_NOT_FOUND	Application group not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

opcapplgrp_get()

```
#include opcsvapi.h

int opcapplgrp_get (
      opc_connection opc_conn,     /* in/out */
      const opcdata  applgrp,      /*     in */
      opcdata        applgrp_conf  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`applgrp`	Application group configuration of type OPCDTYPE_APPL_GROUP.
`applgrp_conf`	Application group configuration of type OPCDTYPE_APPL_GROUP.

Description

Gets the full configuration of the specified application group. The application group must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `applgrp` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`applgrp_conf` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_APPLGROUP_NOT_FOUND`	Application group not found.
`OPC_ERR_LOCKED_BY_OTHER`	This object is currently locked.

OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

opcapplgrp_get_applgrps()

```
#include opcsvapi.h

int opcapplgrp_get_applgrps (
      opc_connection opc_conn,     /* in/out */
      opcdata        applgrp,      /*     in */
      opcdata        applgrp_list  /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
applgrp	Application group configuration of type OPCDTYPE_APPL_GROUP.
applgrp_list	Container of type OPCDTYPE_CONTAINER or OPCDTYPE_APPL_GROUP. The returned elements are of type OPCDTYPE_APPL_GROUP.

Description

Gets a list of all assigned applications groups with the full configuration in the specified application group. If the parent application group is not set (NULL) or the parent UUID is CSMID_C_EMPTY_UUID, the toplevel applications are returned.

The parameter applgrp_list must be an OPCDTYPE_CONTAINER of the type OPCDTYPE_EMPTY or OPCDTYPE_APPL_GROUP.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or applgrp is NULL.
OPC_ERR_INVALID_OUTPARAM	applgrp_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.

OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPLGROUP_NOT_FOUND	Application group not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

opcapplgrp_get_appls()

```
#include opcsvapi.h

int opcapplgrp_get_appls (
      opc_connection opc_conn,  /* in/out */
      opcdata        applgrp,   /*     in */
      opcdata        appl_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`applgrp`	Application group configuration of type OPCDTYPE_APPL_GROUP.
`appl_list`	Container of type OPCDTYPE_CONTAINER or OPCDTYPE_APPL_CONFIG. The returned elements are of type OPCDTYPE_APPL_CONFIG.

Description

Gets a list of all assigned applications with the full configuration in the specified application group. If the parent application group is not set (NULL) or the parent UUID is `CSMID_C_EMPTY_UUID` the toplevel applications are returned.

The parameter `appl_list` must be an OPCDTYPE_CONTAINER of the type OPCDTYPE_EMPTY or OPCDTYPE_APPL_CONFIG.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `applgrp` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`appl_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.

OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPLGROUP_NOT_FOUND	Application group not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_CONTAINER” on page 542

“opc_connect()” on page 181

opcapplgrp_get_list()

```
#include opcsvapi.h

int opcapplgrp_get_list (
      opc_connection opc_conn,     /* in/out */
      opcdata        applgrp_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`applgrp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_APPL_GROUP. The returned elements are of type OPCDTYPE_APPL_GROUP.

Description

Gets a list of all application groups with the full configuration in the application bank.

The parameter `applgrp_list` must be an OPCDTYPE_CONTAINER of the type OPCDTYPE_EMPTY or OPCDTYPE_APPL_GROUP.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `applgrp_list` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`applgrp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_OBJECT_NOT_FOUND`	Application bank not found.
`OPC_ERR_LOCKED_BY_OTHER`	This object is currently locked.
`OPC_ERR_DEADLOCK`	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_CONTAINER” on page 542

“opc_connect()” on page 181

opcapplgrp_modify()

```
#include opcsvapi.h

int opcapplgrp_modify (
      opc_connection opc_conn,     /* in/out */
      const opcdata  applgrp,      /*     in */
      opcdata        applgrp_conf  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`applgrp`	Application group configuration of type OPCDTYPE_APPL_GROUP.
`applgrp_conf`	Application group configuration of type OPCDTYPE_APPL_GROUP.

Description

Modifies the specified application group. The application group must be specified by either the UUID or the name. If the UUID is given, the name will be ignored. The full configuration data will be set.

`applgrp_conf` must contain the full new configuration.

The application group configuration is checked before modification. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition. The name of the application group must be specified.

If an application group with this name already exists, `OPC_ERR_OBJECT_ALREADY_EXISTS` is returned and the application group will not be created.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `applgrp` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`applgrp_conf` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_APPLGROUP_NOT_FOUND	Application group not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“opc_connect()” on page 181

Category Configuration API

The Category Configuration API provides a set of functions for configuration of categories, as well as for management of links between categories and policies or nodes. Category is a concept upon which the related HPOM instrumentation files are configured into a logical unit.

General information about categories is contained in `opcdata` structures of type OPCDTYPE_CATEGORY_INFO.

The Category Configuration API covers the following functionality:

- Adding, modifying, and deleting categories, and getting a list of all categories in the HPOM database.
- Assigning and deassigning categories on a node level.
- Assigning and deassigning categories on a policy level.
- Getting a list of already assigned categories on a policy and a node level.

NOTE

With the exception of `opcinstrum_add_categories()` and `opcinstrum_modify_categories()`, the Category Configuration API does not modify category subdirectory structure in the file system.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Data Structure

The Category Configuration API is used with `opcdata` structures of type OPCDTYPE_CONTAINER, OPCDTYPE_TEMPLATE_INFO, OPCDTYPE_NODE_CONFIG, and OPCDTYPE_CATEGORY_INFO.

OPCDTYPE_CATEGORY_INFO contains a full set of category information. See Table 5-8, “OPCDTYPE_CATEGORY_INFO,” for category configuration attributes.

Return Values

This function returns `OPC_ERR_OK` after successful execution, otherwise the appropriate error code is returned as described in the include file `opcsvapi.h`.

`OPC_ERR_OK`	The execution of the function succeeded and no error occurred.
`OPC_ERR_INVALID_INPARAM`	One of the input parameters is NULL or not of the correct type.
`OPC_ERR_INVALID_OUTPARAM`	One of the input parameters is NULL or not of the correct type.
`OPC_ERR_DATABASE_ERROR`	The access to the database failed.
`OPC_ERR_OBJECT_ALREADY_EXISTS`	The specified object already exists in the database.
`OPC_ERR_INVALID_NODE`	The specified node is either invalid or not accessible.
`OPC_ERR_OBJECT_NOT_FOUND`	The specified node was not found.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_CANT_LOCK_MUTEX`	Global mutex cannot be locked.
`OPC_ERR_NO_MEMORY`	Memory allocation error.
`OPC_ERR_ACCESS_DENIED`	The HPOM user has no administrator rights.

Restrictions

This function can be issued only on the management server.

Multithread Usage

The `opcnode_*()` functions are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads. They are neither async-cancel safe, async-signal, nor fork-safe.

opcinstrum_add_categories()

```
#include opcsvapi.h

int opcinstrum_add_categories (
        const opc_connection   opc_conn,          /* in */
        int                    directory          /* in */
        opcdata                category_list      /* out */
                        );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`directory`	If non-zero, create directory tree for the categories in the file system.
`category_list`	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function `opcinstrum_add_categories()` to add categories to the HPOM database.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL or `categories` is of incorrect type.
`OPC_ERR_OBJECT_ALREADY_EXISTS`	A category with the same ID and name already exists in the database.
`OPC_ERR_DATABASE_ERROR`	The access to the database failed.
`OPC_ERR_NO_MEMORY`	Out of memory.
`OPC_ERR_ACCESS_DENIED`	User has no administrative rights.
`OPC_ERR_CFGUPLD_RUNNING`	Configuration upload in progress.
`OPC_ERR_CANT_LOCK_MUTEX`	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

opcinstrum_del_categories()

```
#include opcsvapi.h

int opcinstrum_del_categories (
        const opc_connection   opc_conn,          /* in */
        int                    force,             /* in */
        opcdata                category_list      /* out */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
force	If non-zero, deletes also node and policy assignments.
category_list	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function opcinstrum_del_categories() to remove categories from the HPOM database.

This call deletes the node and policy category assignments if -force is set to non-zero. If it is not non-zero, then only categories without node or policy assignments are removed.

NOTE Policies lose their category strings in case the assigned categories are deleted using the -force option.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL or categories is of incorrect type.
OPC_ERR_OBJECT_NOT_FOUND	One of the categories does not exist in the database.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

OPC_ERR_NO_MEMORY	Out of memory.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

opcinstrum_get_categories()

```
#include opcsvapi.h

int opcinstrum_get_categories (
        const opc_connection   opc_conn,          /* in */
        opcdata                category_list      /* out */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
category_list	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function opcinstrum_get_categories() to get a list of all categories that exist in the HPOM database.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	categories is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

opcinstrum_get_category()

```
#include opcsvapi.h

int opcinstrum_get_category (
        const opc_connection   opc_conn,          /* in */
        opcdata            category,              /* in */
        opcdata            category_conf          /* out */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
category	Category UUID or the name. Must be an OPCDTYPE_CATEGORY_INFO.
category_conf	A specified category configuration. Must be an OPCDTYPE_CATEGORY_INFO.

Description

Use the function opcinstrum_get_category() to get the full configuration of the specified category. The category must be specified by either the UUID or the name.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	categories is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CATEGORY_INFO” on page 554

opcinstrum_modify_categories()

```
#include opcsvapi.h

int opcinstrum_modify_categories (
        const opc_connection   opc_conn,          /* in */
        opcdata            category_from,         /* in */
        opcdata            category_to            /* out */
        int                update_fs,             /* in */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
category_from	The category to be changed. Must be an OPCDTYPE_CATEGORY_INFO.
category_to	New category name and description. Must be an OPCDTYPE_CATEGORY_INFO.
update_fs	If non-zero, also modify category in the file system.

Description

Use the function opcinstrum_modify_categories() to change name or description of categories in the HPOM database.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL or categories is of incorrect type.
OPC_ERR_OBJECT_NOT_FOUND	One of the categories does not exist in the database.
OPC_ERR_OBJECT_ALREADY_EXISTS	One or more categories cannot be renamed since a category with such name already exists in the database.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CATEGORY_INFO” on page 554

opcpolicy_assign_categories()

```
#include opcsvapi.h

int opcpolicy_assign_categories (
        const opc_connection   opc_conn,          /* in */
        opcdata            policy,            /* in */
        opcdata            category_list          /* out */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
policy	An HPOM policy. Must be an OPCDTYPE_TEMPLATE_INFO with policy ID specified in OPCDATA_ID.
category_list	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function opcpolicy_assign_categories() to assign a list of categories to the specified policy.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn or policy or category_list is NULL or of incorrect type.
OPC_ERR_OBJECT_NOT_FOUND	The specified policy does not exist in the database.
OPC_ERR_OBJECT_ALREADY_EXISTS	A category is already assigned to the policy.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_TEMPLATE_INFO” on page 585

opcpolicy_deassign_categories()

```
#include opcsvapi.h

int opcpolicy_deassign_categories (
        const opc_connection   opc_conn,          /* in */
        opcdata            policy,            /* in */
        opcdata            category_list          /* out */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
policy	An HPOM policy. Must be an OPCDTYPE_TEMPLATE_INFO with policy ID specified in OPCDATA_ID.
category_list	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function opcpolicy_deassign_categories() to deassign a list of categories from the specified policy.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn or policy or category_list is NULL or of incorrect type.
OPC_ERR_OBJECT_NOT_FOUND	The specified policy or one of the categories does not exist in the database.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_TEMPLATE_INFO” on page 585

opcpolicy_get_categories()

```
#include opcsvapi.h

int opcpolicy_get_categories (
        const opc_connection   opc_conn,          /* in */
        opcdata            policy,            /* in */
        opcdata            category_list          /* out */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
policy	An HPOM policy. Must be an OPCDTYPE_TEMPLATE_INFO with policy ID specified in OPCDATA_ID.
category_list	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function opcpolicy_get_categories() to get a list of all categories that are assigned to the specified policy.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	categories is NULL or of incorrect type.
OPC_ERR_OBJECT_NOT_FOUND	The specified policy does not exist in the database.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_TEMPLATE_INFO” on page 585

opcnode_assign_categories()

```
#include opcsvapi.h

int opcnode_assign_categories (
        const opc_connection   opc_conn,          /* in */
        opcdata            managed_node,          /* in */
        opcdata            category_list          /* out */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
managed_node	A managed node. Must be an OPCDTYPE_NODE_CONFIG with node ID specified in OPCDATA_ID.
category_list	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function opcnode_assign_categories() to assign a list of categories to the specified node.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn or node or category_list is NULL or of incorrect type.
OPC_ERR_OBJECT_NOT_FOUND	The specified node does not exist in the database.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_deassign_categories()

```
#include opcsvapi.h

int opcnode_deassign_categories (
        const opc_connection   opc_conn,          /* in */
        opcdata            managed_node,          /* in */
        opcdata            category_list          /* out */
                    );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`managed_node`	A managed node. Must be an OPCDTYPE_NODE_CONFIG with node ID specified in OPCDATA_ID. category_list List of categories. Must be an OPCDTYPE_CONTAINER.
`category_list`	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function `opcnode_deassign_categories()` to deassign a list of categories from the specified node.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` **or** `node` **or** `category_list` is NULL or of incorrect type.
`OPC_ERR_OBJECT_NOT_FOUND`	The specified node does not exist in the database.
`OPC_ERR_DATABASE_ERROR`	The access to the database failed.
`OPC_ERR_NO_MEMORY`	Out of memory.
`OPC_ERR_ACCESS_DENIED`	User has no administrative rights.
`OPC_ERR_CFGUPLD_RUNNING`	Configuration upload in progress.
`OPC_ERR_CANT_LOCK_MUTEX`	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_get_categories()

```
#include opcsvapi.h

int opcnode_get_categories (
        const opc_connection   opc_conn,          /* in */
        opcdata            managed_node,          /* in */
        opcdata            category_list          /* out */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
managed_node	A managed node. Must be an OPCDTYPE_NODE_CONFIG with node ID specified in OPCDATA_ID.
category_list	List of categories. Must be an OPCDTYPE_CONTAINER.

Description

Use the function opcnode_get_categories() to get a list of categories that are assigned to the specified node.

Return Values:

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	categories is NULL or of incorrect type.
OPC_ERR_OBJECT_NOT_FOUND	The specified node does not exist in the database.
OPC_ERR_DATABASE_ERROR	The access to the database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User has no administrative rights.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_CANT_LOCK_MUTEX	Cannot lock mutex for safe execution.

Versions

HPOM 9.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_NODE_CONFIG” on page 576

Instruction Text Interface Configuration API

The Instruction Text Interface Configuration API provides a set of functions to configure the HPOM instruction text interface. You can use the instruction text interface to call an external application to present instructions to an operator.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see “opcdata_create()” on page 49 or “opcdata_clear()” on page 46) and freeing (see “opcdata_free()” on page 51) the needed memory.

Data Structures

OPCDTYPE_INSTR_IF

Usage

To use these functions, it is necessary to connect to the management server as an HPOM user with administrator rights, using the function opc_connect() (see page 181).

1. Use opcdata_create() to create an object of type OPCDTYPE_INSTR_IF.
2. Use opcdata_set_str() to set string attributes to, and opcdata_get_str() to get string attributes from the object of type OPCDTYPE_INSTR_IF.
3. Use opcdata_set_long() to set and opcdata_get_long() to get integer attributes.
4. Call the appropriate opcinstruction_* API functions when all necessary attributes are set or retrieved.
5. Use opcdata_free() to free the object of type OPCDTYPE_INSTR_IF when you no longer need it.

Prerequisites

The Instruction Text Interface API is available only on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcinstruction_add()

```
#include opcsvapi.h

int opcinstruction_add (
      opc_connection  opc_conn,   /* in/out */
      const  opcdata  instrintf   /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
instrintf	Instruction text interface configuration of type OPCDTYPE_INSTR_IF.

Description

Adds a new instruction text interface to the HPOM database.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Instruction text interface already exists.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.27 and later

See Also

“opc_connect()” on page 181

“opcinstruction_del()” on page 251

“opcinstruction_get()” on page 252

“opcinstruction_modify()” on page 254

opcinstruction_del()

```
#include opcsvapi.h

int opcinstruction_del (
      opc_connection  opc_conn,         /* in/out */
      const char *    instruction_name  /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
instruction_name	Instruction text interface name.

Description

Deletes an instruction text interface from the HPOM database.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_OBJECT_NOT_FOUND	Instruction text interface not found.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.25 and later

See Also

“opc_connect()” on page 181

“opcinstruction_add()” on page 249

“opcinstruction_get()” on page 252

“opcinstruction_modify()” on page 254

opcinstruction_get()

```
#include opcsvapi.h

int opcinstruction_get (
      opc_connection  opc_conn,          /* in/out */
      const char *    instruction_name,  /* in  */
      opcdata         instrintf          /* out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`instruction_name`	Instruction text interface name.
`instrintf`	Instruction text interface configuration of type OPCDTYPE_INSTR_IF.

Description

Gets the full configuration of the specified instruction text interface from the HPOM database. The instruction text interface must be specified by the name.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_OBJECT_NOT_FOUND`	Instruction text interface not found.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_CFGUPLD_RUNNING`	Configuration upload in progress.

Versions

8.27 and later

See Also

“opc_connect()” on page 181

“opcinstruction_add()” on page 249

“opcinstruction_del()” on page 251

“opcinstruction_modify()” on page 254

opcinstruction_modify()

```
#include opcsvapi.h

int opcinstruction_modify (
      opc_connection  opc_conn,          /* in/out */
      const char *    instruction_name,  /* in */
      const opcdata   instrintf          /* in */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`instruction_name`	Instruction text interface name.
`instrintf`	Instruction text interface configuration of type OPCDTYPE_INSTR_IF.

Description

Modifies the specified instruction text interface. The instruction text interface must be specified by the name.

`instrintf` must contain the new configuration.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_OBJECT_NOT_FOUND`	Instruction text interface not found.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_CFGUPLD_RUNNING`	Configuration upload in progress.

Versions

8.27 and later

See Also

“opc_connect()” on page 181

“opcinstruction_add()” on page 249

“opcinstruction_del()” on page 251

“opcinstruction_get()” on page 252

Message Group Configuration API

The Message Group Configuration API provides a set of functions to configure HPOM message groups.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see opcdata_create(3) or opcdata_clear(3)) and freeing (see opcdata_free(3)) the needed memory.

Data Structures

OPCDTYPE_MESSAGE_GROUP

A message group is identified by its name `OPCDATA_NAME`.

Usage

To use these functions, it is necessary to connect to the HP Operations management server as an HPOM user with administrator rights, using the function opc_connect() (see page 181). Only the function opcmsggrp_get_list() can be called by any other user.

Prerequisites

The Message Group Configuration API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the Message Group Configuration API are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcmsggrp_add()

```
#include opcsvapi.h

int opcmsggrp_add (
        const opc_connection   opc_conn,          /* in */
        opcdata                msg_group          /* in */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
msg_group	Data structure of type OPCDTYPE_MESSAGE_GROUP.

Description

Use the function opcmsggrp_add() to add a message group to the HPOM database. To add a message group, the message group name OPCDATA_NAME must be specified.

Return Values

If an error occurs, opcmsggrp_add() returns a value within the range OPC_NOTE < error_code < OPC_ERR_OK. If an attribute of the message group is invalid, the function can also return a value greater than zero. The return value specifies the field that was not set, or set incorrectly. For example, OPCDATA_NAME indicates that the name of the message group was not set.

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL msg_group is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_MESSAGE_GROUP” on page 571

opcmsggrp_delete()

```
#include opcsvapi.h

int opcmsggrp_delete (
        const opc_connection  opc_conn,          /* in */
        const opcdata         msg_group          /* in */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
msg_group	Data structure of type OPCDTYPE_MESSAGE_GROUP.

Description

Use the function opcmsggrp_delete() to remove a message group from the HPOM database. To specify the message group, the name must be set.

NOTE Deleting a message group also changes the configuration of the operator who is responsible for that message group. The changes will be visible in GUI immediately.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL msg_group is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_MESSAGE GROUP	Message group is not in the database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_MESSAGE_GROUP” on page 571

opcmsggrp_get()

```
#include opcsvapi.h

int opcmsggrp_get (
        const opc_connection   opc_conn,        /* in */
        const opcdata          msg_group,       /* in */
        opcdata                grp_conf         /* out* /
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
msg_group	Data structure of type OPCDTYPE_MESSAGE_GROUP.
grp_conf	Data structure of type OPCDTYPE_MESSAGE_GROUP.

Description

Use the function opcmsggrp_get() to get the configuration of a given message group. To specify the message group, the name must be set.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL; msg_group is NULL or of incorrect type.
OPC_ERR_INVALID_OUTPARAM	grp_conf is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181
“OPCDTYPE_MESSAGE_GROUP” on page 571

opcmsggrp_get_list()

```
#include opcsvapi.h

int opcmsggrp_get_list (
        const opc_connection   opc_conn,          /* in */
        opcdata                msg_groups         /* out */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
msg_groups	Container of type OPCDTYPE_EMPTY or OPCDTYPE_MESSAGE_GROUP.

Description

Use the function opcmsggrp_get_list() to get a list of all message groups.

If the function is called for an HPOM user other than the administrator, the message groups of this HPOM user will be returned. Otherwise a list of all message groups will be returned.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	msg_groups is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_MESSAGE_GROUP” on page 571

opcmsggrp_modify()

```
#include opcsvapi.h

int opcmsggrp_modify (
        const opc_connection  opc_conn,          /* in */
        const opcdata         msg_group,         /* in */
        const opcdata         mod_msg_group      /* in */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
msg_group	Data structure of type OPCDTYPE_MESSAGE_GROUP.
mod_msg_group	Modified message group of type OPCDTYPE_MESSAGE_GROUP.

Description

Use the function opcmsggrp_modify() to modify the description and symbol of a message group. (The name of the message group cannot be changed.) To specify the message group, the name must be given.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL msg_group is NULL mod_msg_group is NULL or of incorrect type.
OPC_ERR_INCOMPLETE_INPARAM	Modified message group configuration is incomplete.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_MESSAGE GROUP	Message group is not in the database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_MESSAGE_GROUP” on page 571

Message Regroup Condition Configuration API

The message regroup condition API provides a set of functions to configure HPOM message regroup conditions.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see opcdata_create(3) or opcdata_clear(3)) and freeing (see opcdata_free(3)) the needed memory.

Data Structures

`OPCDTYPE_REGROUP_COND`

A message regroup condition is specified either by name or by the UUID. If the UUID is given, a specified name will be ignored.

Usage

To use these functions, it is necessary to connect to the HP Operations management server as an HPOM user with administrator rights, using the function opc_connect() (see page 181).

Prerequisites

The Message Regroup Condition Configuration API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcmsgregrp_add()

```
#include opcsvapi.h

int opcmsgregrp_add (
      opc_connection opc_conn, /* in/out */
      opcdata        mcond     /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
mcond	Message regroup conditions to add; data structure of the type OPCDTYPE_REGROUP_COND.

Description

Adds the specified message regroup condition.

The regroup condition configuration is checked before creation. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

The name of the condition must be specified. If a condition with this name already exists, OPC_ERR_OBJECT_ALREADY_EXISTS is returned and the condition will not be created.

The condition will be appended to the end of the list of conditions. If creation was successful, the UUID of the new condition will be set in mcond.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	mcond is NULL or invalid type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.

OPC_ERR_OBJECT_ALREADY_EXISTS	Condition already exists.

Versions

5.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 584

“opc_connect()” on page 181

“opcdata_create()” on page 49

opcmsgregrp_delete()

```
#include opcsvapi.h

int opcmsgregrp_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        mcond     /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
mcond	Message regroup condition to be deleted; data structure of the type OPCDTYPE_REGROUP_COND.

Description

Deletes the specified message regroup condition. The condition must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The regroup condition configuration is checked before deletion. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

If the specified condition does not exist OPC_ERR_OBJECT_NOT_FOUND is returned.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or mcond is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_NOT_FOUND	Condition not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 584

“opc_connect()” on page 181

opcmsgregrp_get()

```
#include opcsvapi.h

int opcmsgregrp_get (
      opc_connection opc_conn,    /* in/out */
      opcdata        mcond,       /*     in */
      opcdata        mcond_conf   /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`mcond`	Message regroup condition to be returned, of type OPCDTYPE_REGROUP_COND.
`mcond_conf`	Configuration of matching condition, of type OPCDTYPE_REGROUP_COND.

Description

Gets the full configuration of the specified message regroup condition. The condition must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `mcond` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`mcond_conf` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.
`OPC_ERR_OBJECT_NOT_FOUND`	Condition not found.
`OPC_ERR_LOCKED_BY_OTHER`	This object is currently locked.
`OPC_ERR_DEADLOCK`	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 584

“opc_connect()” on page 181

opcmsgregrp_get_list()

```
#include opcsvapi.h

int opcmsgregrp_get_list (
      opc_connection opc_conn,    /* in/out */
      opcdata        mcond_list   /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
mcond_list	Ordered list of all conditions will be returned. mcond_list must be an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_REGROUP_COND.

Description

Gets a list of all message regroup conditions with the full configuration of each single message regroup condition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_OUTPARAM	mcond_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_NOT_FOUND	Condition not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_REGROUP_COND” on page 584

“opc_connect()” on page 181

opcmsgregrp_modify()

```
#include opcsvapi.h

int opcmsgregrp_modify (
      opc_connection opc_conn,    /* in/out */
      opcdata        mcond,       /*     in */
      opcdata        mcond_conf   /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
mcond	Message regroup condition to be modified; data structure of type OPCDTYPE_REGROUP_COND.
mcond_conf	Modified condition data of type, data structure of type OPCDTYPE_REGROUP_COND.

Description

Modifies the specified message regroup condition. The condition must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The regroup condition configuration is checked before modification. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

If the name of the condition should be changed and if a condition with this name already exists, OPC_ERR_OBJECT_ALREADY_EXISTS is returned and the condition will not be modified.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or mcond is NULL.
OPC_ERR_INVALID_OUTPARAM	mcond_conf is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.

OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_NOT_FOUND	Condition not found.
OPC_ERR_OBJECT_ALREADY_EXISTS	Condition already exists.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 584

“opc_connect()” on page 181

opcmsgregrp_move()

```
#include opcsvapi.h

int opcmsgregrp_move (
      opc_connection opc_conn, /* in/out */
      opcdata        mcond,    /* in/out */
      unsigned int   to_pos    /*     in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
mcond	Message regroup condition to be moved, data structure of type OPCDTYPE_REGROUP_COND.
to_pos	New position in the condition list. The new position must be between 1 and LAST_ELEMENT; other positions generate an error. (The list starts with the number 1.)

Description

Moves the specified message regroup condition to the specified position in the list of message regroup conditions The condition must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If the specified condition does not exist OPC_ERR_OBJECT_NOT_FOUND is returned.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or mcond is NULL or to_pos is out of range.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_NOT_FOUND	Condition not found.

OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 584

“opc_connect()” on page 181

Node Configuration API

The Node Configuration API provides a set of functions to configure HP Operations managed nodes and node groups. In addition, it allows to add, modify, and delete managed nodes and node groups, and to change the attributes of a node.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see “opcdata_create()” on page 49 or “opcdata_clear()” on page 46) and freeing (see “opcdata_free()” on page 51) the required memory.

Data Structures

The Node Configuration API works with the following types of opcdata structures:

OPCDTYPE_NODE	Contains a minimal set of node information, for example, IP address, network type, and machine type. This is a subset of the OPCDTYPE_NODE_CONFIG structure and is mainly used to get an overview of all nodes, or to specify a node for modification.
OPCDTYPE_NODE_CONFIG	Contains the full set of HP Operations node attributes. For an overview of all node configuration attributes, see Table 5-19, “OPCDTYPE_NODE_CONFIG,” on page 576.

Except for the functions opcnode_get_list() and opcnode_add(), the node is specified by the node name `OPCDATA_NAME` and its network type `OPCDATA_NETWORK_TYPE`, or its IP address `OPCDATA_IP_ADDRESS`, or its UUID `OPCDATA_ID`. The node UUID has the highest priority. When the node UUID is set, the name, the network type, and the IP address are ignored.

OPCDTYPE_NODE_GROUP	Contains the full set of HP Operations node group attributes. See Table 5-20, “OPCDTYPE_NODE_GROUP,” on page 580.

Except for the functions opcnodegrp_get_list() and opcnodegrp_add(), the node group is accessed using its UUID (`OPCDATA_ID`), if set. Otherwise the node group is looked up by name (`OPCDATA_NAME`).

Usage

To use these functions, it is necessary to connect to the management server as administrator, using the function opc_connect() (see page 181). This does not apply to the opcnode_get_list() function.

Prerequisites

The functions are only available on the HP Operations management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcnode_add()

```
#include opcsvapi.h


int opcnode_add (
     const opc_connection   opc_conn,            /* in */
     opcdata                node_conf            /* in */
                 );
```

Parameters

opc_conn	Connection to the HPOM database.
node_conf	Complete node configuration of type OPCDTYPE_NODE_CONFIG.

Description

Use the function opcnode_add() to add a given managed node to the HPOM Node Bank. Initially, the new node is placed in the top level of the node hierarchy; use the function opcnodegrp_assign_nodes() to assign it to another node group.

You can fill the node configuration structure with data using the opcdata API, see “Data API” on page 41. It may, however, be useful to retrieve default values for the new node using the function opcnode_get_defaults(). See “opcnode_get_defaults()” on page 290 for more information.

Nodes with a network type of OP_NETWORK_IP must be known to the name service. See also *gethostbyname(3)* and *gethostbyaddr(3)*.

Return Values

This function can return a positive value if a node configuration attribute is set incorrectly. A positive return value means that a field in the node configuration data structure is set with an improper value. The return value is the value of the opcdata field differentiator of the field that contains the wrong value.

`OPC_DATA_...`	Field differentiator of field with improper value.
`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `node_conf` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Out of memory.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_ALREADY_EXISTS`	Node is already in the database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opcnode_get_defaults()” on page 290

“opcnodegrp_assign_nodes()” on page 301

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_assign_templates()

```
#include opcsvapi.h

int opcnode_assign_templates (
        const opc_connection   opc_conn,          /* in */
        opcdata                mgd_node,          /* in */
        opcdata                policies          /* in */
                              );
```

Parameters:

opc_conn	Connection to the HPOM database.
mgd_node	Data structure of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.
policies	Description of HPOM policies; container of type OPCDTYPE_TEMPLATE_INFO.

Description

The function opccnode_assign_templates() is used for backward compatibility only. Use this function to assign policies to a managed node. A node is identified by its name or its UUID; the UUID has higher priority.

Each policy must be of the type OPCDTYPE_TEMPLATE_INFO, and an opcdata container of the same type must be used. A policy is described by its UUID or its name and type; the UUID has higher priority.

If a specified policy is already assigned to the node, the assignment will be ignored and a warning will be returned.

The OPCDATA_STATUS field of the policy information will be set to OPC_ERR_ALREADY_DONE.

Note that the function only works with opcdata policies and not with policy files.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL mgd_node is NULL policies is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.

OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_NODE	Managed node is not in the database.
OPC_ERR_NOT_COMPLETELY_DONE	Not all policies were assigned to the node.
OPC_WARN_EMPTY_CONTAINER	No policies were specified in policies.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_deassign_templates()

```
#include opcsvapi.h

int opcnode_deassign_templates (
        const opc_connection   opc_conn,          /* in */
        opcdata                mgd_node,          /* in */
        opcdata                policies          /* in */
                                );
```

Parameters

opc_conn	Connection to the HPOM database.
mgd_node	Data structure of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.
policies	Description of HPOM policies; this is a container of type OPCDTYPE_TEMPLATE_INFO.

Description

The function opccnode_deassign_templates() is used for backward compatibility only. Use this function to deassign policies from a managed node. A node is identified by its name or its UUID; the UUID has higher priority.

Each policy must be of the type OPCDTYPE_TEMPLATE_INFO, and an opcdata container of the same type must be used. A policy is described by its UUID or its name and type; the UUID has higher priority.

If a specified policy has already been deassigned from the node the deassignment will be ignored and a warning will be returned.

The OPCDATA_STATUS field of the policy information will be set to OPC_ERR_ALREADY_DONE.

Note that the function only works with opcdata policies and not with policy files.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL mgd_node is NULL policies is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.

OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_NODE	Managed node is not in the database.
OPC_ERR_NOT_COMPLETELY_DONE	Not all policies were deassigned.
OPC_WARN_EMPTY_CONTAINER	No policies were specified in policies.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_delete()

```
#include opcsvapi.h

int opcnode_delete (
         const opc_connection   opc_conn,          /* in */
         const opcdata          mgd_node           /* in */
                  );
```

Parameters

opc_conn	Connection to the HPOM database.
mgd_node	Data structure of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.

Description

opccnode_delete() deletes the specified node from the HPOM database and removes all assignments to node groups and node layout groups. It also acknowledges all messages originating from the node in the database.

If the node is referenced in a message source policy, application, or message, a warning is issued. Generate the Node Reference Report to identify all references.

A node is identified by its name or its UUID; the UUID has higher priority.

NOTE Deleting a node from the HPOM database also changes the configuration of the operators who are responsible for that node. This function does not automatically update any running operator GUIs.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL mgd_node is NULL or is of incorrect type.
OPC_ERR_INCOMPLETE_INPARAM	Definition of node is incomplete.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.

OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_NODE	Managed node not in database.
OPC_WARN_NODE_IS_REFERENCED	opc_node_names entry remains in database because node is still referenced in policies, applications, or messages.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_get()

```
#include opcsvapi.h

int opcnode_get (
        const opc_connection   opc_conn,          /* in */
        const opcdata          mgd_node,          /* in */
        opcdata                node_conf          /* out */
                );
```

Parameters:

opc_conn	Connection to the HPOM database.
mgd_node	Node of type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.
node_conf	Node configuration of type OPCDTYPE_NODE_CONFIG or OPCDTYPE_EMPTY.

Description

Use the function opcnode_get() to get information about a given managed node from the HPOM database. A node is identified by its name or its UUID; the UUID has higher priority.

The returned data structure of the type OPCDTYPE_NODE_CONFIG contains all necessary information to define a node in HPOM. This function could be used as basis for changing node attributes or adding a new node with similar parameters.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL mgd_node is NULL or is of an incorrect type.
OPC_ERR_INVALID_OUTPARAM	node_conf is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

OPC_ERR_INVALID_NODE	Managed node is not in the database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_get_defaults()

```
#include opcsvapi.h

int opcnode_get_defaults (
          const opc_connection   opc_conn,          /* in */
          const opcdata          mgd_node,          /* in */
          opcdata                node_conf          /* out */
                          );
```

Parameters:

opc_conn	Connection to the HPOM database.
mgd_node	Node of type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.
node_conf	Node configuration of type OPCDTYPE_NODE_CONFIG.

opcnode_get_defaults() does not overwrite the node name OPCDATA_NAME, the IP address OPCDATA_IP_ADDRESS, or the UUID OPCDATA_ID in the node configuration structure.

Description

Use the function opcnode_get_defaults() to fill an empty OPCDTYPE_NODE_CONFIG structure with default values depending on the given network and machine type. Specify the node type with network OPCDATA_NETWORK_TYPE and machine type OPCDATA_MACHINE_TYPE.

opcnode_get_defaults() does not overwrite the node name OPCDATA_NAME, the IP address OPCDATA_IP_ADDRESS, or the UUID OPCDATA_ID in the node configuration structure.

Once the structure has been filled with the default values, you only need to add the name and description, and the parameters you want to modify to define the new node.

Create empty structures with the function opcdata_create().

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL mgd_node is NULL.
OPC_ERR_INVALID_OUTPARAM	node_conf is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_NODE_TYPE	Invalid network machine type.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opcdata_create()” on page 49

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_get_list()

```
#include opcsvapi.h

int opcnode_get_list (
         const opc_connection   opc_conn,          /* in */
         opcdata                nodes              /* out */
                     );
```

Parameters

opc_conn	Connection to the HPOM database.
nodes	Container of type OPCDTYPE_EMPTY or OPCDTYPE_NODE.

Prototype

Use the function opcnode_get_list() to get a list of nodes assigned to the operator specified in opc_conn. If the connection is established as administrator, a list of all nodes is returned. The returned container is a list of type OPCDTYPE_NODE.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	container is NULL or is of incorrect type.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE” on page 574

opcnode_get_templates()

```
#include opcsvapi.h

int opcnode_get_templates (
         const opc_connection   opc_conn,          /* in */
         const opcdata          mgd_node,          /* in */
         opcdata                policies          /* out */
                          );
```

Parameters

opc_conn	Connection to the HPOM database.
mgd_node	Data structure of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.
policies	Container of type OPCDTYPE_TEMPLATE_INFO.

Description

The function opccnode_get_templates() is used for backward compatibility only. This function returns a list of all policies and policy groups that are assigned to the managed node. A node is identified by its name or its UUID; the UUID has higher priority.

The policy list is returned in a container of the type OPCDTYPE_TEMPLATE_INFO. This container element does not contain a complete definition of the policy, only the policy name, policy type and policy UUID. You can use these elements and the policy file API to get a full description of the policy. The container can also contain policy groups which can be identified by the policy type.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL mgd_node is NULL policies is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

OPC_ERR_INVALID_NODE	Managed node is not in the database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_modify()

```
#include opcsvapi.h

int opcnode_modify (
        const opc_connection   opc_conn,          /* in */
        const opcdata          mgd_node,          /* in */
        const opcdata          new_node_conf      /* in */
                    );
```

Parameters:

`opc_conn` Connection to the HPOM database.

`mgd_node` Data structure of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.

`new_node_conf` New and complete node configuration of type OPCDTYPE_NODE_CONFIG. All attributes must be set to valid values. This is guaranteed by using opcnode_get() or opcnode_get_defaults(). The IP address of the node is updated if it differs in `mgd_node` and `new_node_conf`. Only following network type modifications are allowed:

`OPC_NETWORK_IP` to `OPC_NETWORK_OTHER`

`OPC_NETWORK_OTHER` to `OPC_NETWORK_IP`

`OPC_NODE_PATTERN_IP_ADDR` to `OPC_NODE_PATTERN_IP_NAME`

`OPC_NODE_PATTERN_IP_ADDR` to `OPC_NODE_PATTERN_OTHER`

`OPC_NODE_PATTERN_IP_NAME` to `OPC_NODE_PATTERN_IP_ADDR`

`OPC_NODE_PATTERN_IP_NAME` to `OPC_NODE_PATTERN_OTHER`

`OPC_NODE_PATTERN_IP_OTHER` to `OPC_NODE_PATTERN_IP_ADDR`

`OPC_NODE_PATTERN_IP_OTHER` to `OPC_NODE_PATTERN_IP_NAME`

Description

Use the function `opcnode_modify()` to change the attributes of an already existing node in the HPOM database. A node is identified by its name or its UUID; the UUID has higher priority. The modifications are done in the node configuration structure using the opcdata API, see “Data API” on page 41.

To retrieve a list of managed nodes, use the function `opcnode_get_list()`, see “opcnode_get_list()” on page 292. This function returns a container of type `OPCDTYPE_MANAGED_NODE` which allows you to specify the nodes that you want to change. Stepping through this list allows mass operations on a large number of managed nodes.

Return Values

This function can return a positive value which means that a field in the node configuration data structure is set with an improper value. The return value is the value of the opcdata field differentiator of the field that contains the wrong value.

`OPC_DATA_...`	Field differentiator of field with improper value.
`OPC_ERR_OK`	OK
`OPC_ERR_INCOMPLETE_INPARAM`	Node configuration is incomplete, node is incomplete.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Out of memory.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_INVALID_NODE`	Managed node is not in the database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE” on page 574
“OPCDTYPE_NODE_CONFIG” on page 576

opcnode_modify_defaults()

```
#include opcsvapi.h

int opcnode_modify_defaults (
      opc_connection  opc_conn,      /* in */
      opcdata         node_conf      /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
node_conf	Node configuration of type OPCDTYPE_NODE_CONFIG.

Description

Use the function opcnode_modify_defaults() to change the default values of the given network and machine type.

To retrieve default values depending on the given network and machine type, use the function opcnode_get_defaults(). This function fills the node configuration structure with the default values from the HPOM database.

After retrieving the defaults values, modify the node configuration data structure using the functions of the Data API, see “Data API” on page 41.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.
OPC_ERR_INVALID_NODE_TYPE	Invalid network or machine type.

Versions

8.25 and later

See Also

“opc_connect()” on page 181

“opcnode_get_defaults()” on page 290

opcnodegrp_add()

```
#include opcsvapi.h

int opcnodegrp_add (
        const opc_connection   opc_conn,          /* in */
        opcdata                node_group         /* in */
                    );
```

Parameters:

opc_conn	Connection to the HPOM database.
node_group	Node group definition of the type OPCDTYPE_NODE_GROUP.

Description

Use the function opcnodegrp_add() to add a new node group to the HPOM database. A node group is specified by its name. After successful completion, the node group UUID is returned in the field OPCDATA_ID of node_group.

Initially, there is no node assigned to the new node group. Use the function opcnodegrp_assign_nodes() to assign nodes to your new node group. See “opcnodegrp_assign_nodes()” on page 301 for more information.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL node_group is NULL or is of incorrect type.
OPC_ERR_INCOMPLETE_INPARAM	Node group configuration is incomplete.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opcnodegrp_assign_nodes()” on page 301

“OPCDTYPE_NODE_GROUP” on page 580

opcnodegrp_assign_nodes()

```
#include opcsvapi.h

int opcnodegrp_assign_nodes (
        const opc_connection   opc_conn,          /* in */
        opcdata                node_group,        /* in */
        opcdata                nodes              /* in */
                              );
```

Parameters

opc_conn	Connection to the HPOM database.
node_group	Node group to be assigned; opcdata structure of type OPCDTYPE_NODE_GROUP.
nodes	Container of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.

Description

Use the function opcnodegrp_assign_nodes() to assign nodes to a node group. A node group is identified by its name or its UUID; the UUID has higher priority.

If a node has already been assigned to a node group the assignment will be ignored and an appropriate value will be returned in the OPCDATA_STATUS field of the object.

NOTE Assigning a node to a node group also changes the configuration of the operator who is responsible for that node group. The changes will be visible in GUI immediately.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL node_group is NULL or of incorrect type nodes is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.

OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_NODE_GROUP	Node group is not in the database.
OPC_ERR_NOT_COMPLETELY_DONE	Not all nodes were assigned to the node group.
OPC_WARN_EMPTY_CONTAINER	No nodes were specified in nodes.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODE_CONFIG” on page 576

opcnodegrp_assign_templates()

```
#include opcsvapi.h

int opcnodegrp_assign_templates (
        const opc_connection   opc_conn,          /* in */
        const opcdata          node_group,        /* in */
        const opcdata          policies          /* in */
                                 );
```

Parameters

opc_conn	Connection to the HPOM database.
node_group	Node group to which the policies are to be assigned; data structure of type OPCDTYPE_NODE_GROUP.
policies	Container of type OPCDTYPE_TEMPLATE_INFO.

Description

The function opcnodegrp_assign_templates() is used for backward compatibility only. Use this function to assign policies to a node group. When you assign policies to a node group, the policies are automatically assigned to *all* nodes in the node group.

A node group is identified by its name or its UUID; the UUID has higher priority.

Each policy must be of the type OPCDTYPE_TEMPLATE_INFO, and an opcdata container of the same type must be used. A policy is described by its UUID or its name and type; the UUID has higher priority.

If a policy is already assigned to the node group the assignment will be ignored and an appropriate value will be returned in the OPCDATA_STATUS field.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL node_group is NULL or of incorrect type policies is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.

OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_NODE_GROUP	Node group is not in the database.
OPC_ERR_NOT_COMPLETELY_DONE	Not all policies were assigned to the node (group).
OPC_WARN_EMPTY_CONTAINER	No policies were specified in policies.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

“OPCDTYPE_TEMPLATE_INFO” on page 585

opcnodegrp_deassign_nodes()

```
#include opcsvapi.h

int opcnodegrp_deassign_nodes (
       const opc_connection   opc_conn,         /* in */
       opcdata                node_group,       /* in */
       opcdata                nodes             /* in */
                               );
```

Parameters

opc_conn	Connection to the HPOM database.
node_group	Node group from which the nodes are to be deassigned; data structure of type OPCDTYPE_NODE_GROUP.
nodes	Container of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG.

Description

Use the function opcnodegrp_deassign_nodes() to deassign nodes from a node group. A node group is identified by its name or its UUID; the UUID has higher priority.

A node is identified by its name or its UUID; the UUID has higher priority.

If a node has already been deassigned from a node group the deassignment will be ignored and an appropriate value will be returned in the OPCDATA_STATUS field of the object.

NOTE Deassigning a node from a node group also changes the configuration of the operator who is responsible for that node group. The changes will be visible in GUI immediately.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL node_group is NULL or of incorrect type nodes is NULL or of incorrect type.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while `opccfgupld` is running.
OPC_ERR_INVALID_NODE_GROUP	Node group is not in the database.
OPC_ERR_NOT_COMPLETELY_DONE	Not all nodes were deassigned from the node group.
OPC_WARN_EMPTY_CONTAINER	No nodes were specified in nodes.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODE_CONFIG” on page 576

opcnodegrp_deassign_templates()

```
#include opcsvapi.h

int opcnodegrp_deassign_templates (
       const opc_connection   opc_conn,         /* in */
       const opcdata          node_group,       /* in */
       const opcdata          policies         /* in */
                                    );
```

Parameters

opc_conn	Connection to the HPOM database.
node_group	Node group from which the policies are to be deassigned; data structure of type OPCDTYPE_NODE_GROUP.
policies	Container of type OPCDTYPE_TEMPLATE_INFO.

Description

The function opcnodegrp_deassign_templates() is used for backward compatibility only. Use this function to deassign policies from a node group. A node group is identified by its name or its UUID; the UUID has higher priority.

Each policy must be of the type OPCDTYPE_TEMPLATE_INFO, and an opcdata container of the same type must be used. A policy is described by its UUID or its name and type; the UUID has higher priority.

If a policy is already deassigned from the node group the deassignment will be ignored and an appropriate value will be returned in the OPCDATA_STATUS field.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL node_group is NULL or of incorrect type policies is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_NODE_GROUP	Node group is not in the database.
OPC_ERR_NOT_COMPLETELY_DONE	Not all policies were deassigned from the node group.
OPC_WARN_EMPTY_CONTAINER	No policies were specified in policies.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

“OPCDTYPE_TEMPLATE_INFO” on page 585

opcnodegrp_delete()

```
#include opcsvapi.h

int opcnodegrp_delete (
        const opc_connection   opc_conn,         /* in */
        const opcdata          node_group        /* in */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
node_group	Node group to be deleted; opcdata structure of type OPCDTYPE_NODE_GROUP.

Description

Use the function opcnodegrp_delete() to delete a node group from the HPOM database and remove all node assignments. The following data will be updated accordingly:

- Policy assignment
- Nodes assignment
- Operator responsibilities
- Operator layouts

A node group is identified by its name or its UUID; the UUID has higher priority.

NOTE Deleting a node group from the HPOM database also changes the configuration of the operators who are responsible for that node. The changes will be visible in GUI immediately.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL node_group is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.

OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_INVALID_NODE_GROUP	Node group is not in the database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

opcnodegrp_get()

```
#include opcsvapi.h

int opcnodegrp_get (
       const opc_connection   opc_conn,         /* in */
       const opcdata          node_group,       /* in */
       opcdata                nodegrp_conf      /* out */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
node_group	Node group specification of the type OPCDTYPE_NODE_GROUP.
nodegrp_conf	Data structure of the type OPCDTYPE_NODE_GROUP.

Description

Use the function opcnodegrp_get() when you want to get the complete definition of a node group but only know the name or UUID of the group.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL node_group is NULL nodegrp_conf is not of type OPCDTYPE_NODE_GROUP.
OPC_ERR_INVALID_OUTPARAM	nodegrp_conf is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

opcnodegrp_get_list()

```
#include opcsvapi.h

int opcnodegrp_get_list (
       const opc_connection   opc_conn,          /* in */
       opcdata                container          /* out */
                          );
```

Parameters

opc_conn	Connection to the HPOM database.
container	Container of the type OPCDTYPE_EMPTY or OPCDTYPE_NODE_GROUP.

Description

Use the function opcnodegrp_get_list() to get a list of all defined node groups. The returned container contains elements of the type OPCDTYPE_NODE_GROUP.

You can use the list to change the assignment of a node to a node group using the functions opcnodegrp_assign_node() or opcnodegrp_deassign_node(). See “opcnodegrp_assign_nodes()” on page 301 and “opcnodegrp_deassign_nodes()” on page 305 for more information. If the container already contains elements, the node group is added.

The list does not, however, include the nodes assigned to a node group. Use the function opcnodegrp_get_nodes() to get details of the node to node group assignments. See “opcnodegrp_get_nodes()” on page 315 for more information.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	container is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

opcnodegrp_get_nodes()

```
#include opcsvapi.h

int opcnodegrp_get_nodes (
        const opc_connection   opc_conn,        /* in */
        const opcdata          node_group,      /* in */
        opcdata                nodes            /* out */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
node_group	Node group in opcdata structure of type OPCDTYPE_NODE_GROUP.
nodes	Container of the type OPCDTYPE_NODE.

Description

Use the function opcnodegrp_get_nodes() to get a list of all managed nodes assigned to a node group. A node group is identified by its name or its UUID; the UUID has higher priority.

The function returns an opcdata structure of the type OPCDTYPE_NODE. This list can be used to change the attributes of all nodes in a node group using the functions of the Node Configuration API. See “Node Configuration API” on page 278 for more information.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL node_group is NULL or of incorrect type nodes is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Out of memory.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_INVALID_NODE_GROUP	Node group is not in the database.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

“OPCDTYPE_NODE” on page 574

opcnodegrp_get_templates()

```
#include opcsvapi.h

int opcnodegrp_get_templates (
       const opc_connection   opc_conn,         /* in */
       const opcdata          node_group,       /* in */
       const opcdata          policies         /* out */
                              );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`node_group`	Data structure of type OPCDTYPE_NODE_GROUP.
`policies`	Container of type OPCDTYPE_TEMPLATE_INFO.

Description

The function `opcnodegrp_get_templates()` is used for backward compatibility only. Use this function to get a list of the policies and policy groups assigned to a node group. A node group is identified by its name or its UUID; the UUID has higher priority.

The policy list could be used to change the policy to node group assignments using the functions `opcnodegroup_assign_templates()` and `opcnodegroup_deassign_templates()`. See “opcnodegrp_assign_templates()” on page 303 and “opcnodegrp_deassign_nodes()” on page 305 for more information.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `node_group` is NULL or of incorrect type `policies` is NULL or of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Out of memory.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_INVALID_NODE_GROUP`	Node group is not in the database.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

“OPCDTYPE_TEMPLATE_INFO” on page 585

opcnodegrp_modify()

```
#include opcapi.h

int opcnodegrp_modify (
       const opc_connection   opc_conn,         /* in */
       const opcdata          node_group,       /* in */
       const opcdata          mod_node_group    /* in */
                        );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`node_group`	Node group to be modified; opcdata structure of type OPCDTYPE_NODE_GROUP.
`mod_node_group`	New node group configuration of the type OPCDTYPE_NODE_GROUP.

Description

Use the function `opcnodegrp_modify()` to change the attributes of a node group, for example the name and the description. The node group UUID *cannot* be changed. A node group is identified by its name or its UUID; the UUID has higher priority.

Note that the node to node group assignment can only be changed with the functions opcnodegrp_assign_nodes() and opcnodegrp_deassign_nodes(). See “opcnodegrp_assign_nodes()” on page 301 and “opcnodegrp_deassign_nodes()” on page 305 for more information.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `node_group` is NULL or of incorrect type `mod_node_group` is NULL or of incorrect type.
`OPC_ERR_INCOMPLETE_INPARAM`	Modified node group configuration is incomplete.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Out of memory.

OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_INVALID_NODE_GROUP	Node group is not in the database.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_NODE_GROUP” on page 580

Node Hierarchy Configuration API

The Node Hierarchy Configuration API provides a set of functions to configure HP Operations node hierarchies.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see *opcdata_create(3)* or *opcdata_clear(3)*) and freeing (see *opcdata_free(3)*) the needed memory.

Data Structures

OPCDTYPE_NODEHIER	A node hierarchy is specified either by name or by the UUID. If the UUID is given, a specified name will be ignored.
OPCDTYPE_LAYOUT_GROUP	A node layout group is specified either by name or by the UUID. If the UUID is given, a specified name will be ignored.
OPCDTYPE_NODE	Contains a minimal set of node information, for example, IP address, network type, and machine type. This is a subset of the OPCDTYPE_NODE_CONFIG structure and is mainly used to get an overview of all nodes, or to specify a node for modification. A node is specified by the node name `OPCDATA_NAME` and its network type `OPCDATA_NETWORK_TYPE`, or its IP address `OPCDATA_IP_ADDRESS`, or its UUID `OPCDATA_ID`. The node UUID has the highest priority. When the node UUID is set, the name, the network type, and the IP address are ignored.

Usage

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect(). See the man page *opc_connect(3)* for more information.

Prerequisites

The Node Hierarchy Configuration API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcnodehier_add()

```
#include opcsvapi.h

int opcnodehier_add (
      opc_connection opc_conn, /* in/out */
      opcdata        nodehier  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
nodehier	Node hierarchy configuration of type OPCDTYPE_NODEHIER.

Description

Adds the specified node hierarchy.

The node hierarchy configuration is checked, before creating. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition. The name of the node hierarchy must be specified.

If a hierarchy with this name already exists, OPC_ERR_OBJECT_ALREADY_EXISTS is returned and the node hierarchy will not be created.

The UUID of the created object will be returned in the opcdata structure if successful.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	nodehier is NULL or invalid type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.

OPC_ERR_OBJECT_ALREADY_EXISTS	Hierarchy name already exists.

Versions

5.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

“opcdata_create()” on page 49

opcnodehier_add_layoutgrp()

```
#include opcsvapi.h

int opcnodehier_add_layoutgrp (
      opc_connection   opc_conn,         /* in/out */
      const opcdata    nodehier,         /*     in */
      const opcdata    parentlayoutgrp,  /*     in */
      opcdata          layoutgrp         /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
`parentlayoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.
`layoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.

Description

Adds the specified layout group to the specified node hierarchy. A node layout group and a node hierarchy are specified either by name or by the UUID. If the UUID is given, a specified name will be ignored.

The parent layout group defines the location for this layout group. If the parent layout group is NULL, the layout group will be created at the toplevel of the node hierarchy. The parent layout group can be specified by either UUID or name and path. The UUID will supersede the name if both are given. If you want to create a nested layout group, you must specify the complete path from the toplevel layout group using the `OPCDATA_PATH` field of the `OPCDTYPE_LAYOUT_GROUP` data structure. Each level must be separated by a slash, for example `level-1/level-2`.

The layout group configuration is checked before creating. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

If a layout group with this name already exists, `OPC_ERR_OBJECT_ALREADY_EXISTS` is returned and the layout group will not be created.

The UUID of the created object will be returned in the opcdata structure if successful.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	nodehier is NULL or invalid type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Layout group already exists.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Parent not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

“opcdata_create()” on page 49

opcnodehier_copy()

```
#include opcsvapi.h

int opcnodehier_copy (
      opc_connection   opc_conn,     /* in/out */
      const opcdata    src_nodehier, /*     in */
      opcdata          dst_nodehier  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`src_nodehier`	Source node hierarchy configuration of type OPCDTYPE_NODEHIER.
`dst_nodehier`	Destination node hierarchy configuration of type OPCDTYPE_NODEHIER.

Description

Copies the specified node hierarchy.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored. The full configuration data will be set.

The node hierarchy configuration is checked, before copying. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition. The name of the node hierarchy must be specified.

If a hierarchy with this name already exists, `OPC_ERR_OBJECT_ALREADY_EXISTS` is returned and the node hierarchy will not be created.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or src_nodehier is NULL.
OPC_ERR_INVALID_OUTPARAM	dst_nodehier is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_delete()

```
#include opcsvapi.h

int opcnodehier_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        nodehier  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
nodehier	Node hierarchy configuration of type OPCDTYPE_NODEHIER.

Description

Deletes the specified node hierarchy.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or nodehier is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 581
“opc_connect()” on page 181

opcnodehier_delete_layoutgrp()

```
#include opcsvapi.h

int opcnodehier_delete_layoutgrp (
      opc_connection opc_conn,  /* in/out */
      const opcdata  nodehier,  /*     in */
      opcdata        layoutgrp  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
`layoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.

Description

Deletes the specified layout group. The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The layout group can be specified by either UUID or name and path. The UUID will supersede the name if both are given.

If you want to delete a nested layout group, you must specify the complete path from the toplevel layout group using the `OPCDATA_PATH` field of the `OPCDTYPE_LAYOUT_GROUP` data structure. Each level must be separated by a slash, for example `level-1/level-2`.

The layout group configuration is checked, before deleting. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Parent not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_LAYOUTGROUP_NOT_EMPTY	Attempt to delete not empty layout group.
OPC_ERR_LAYOUTGROUP_IS_HOLDING_AREA	Attempt to delete the Holding Area.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_get()

```
#include opcsvapi.h

int opcnodehier_get (
      opc_connection opc_conn,      /* in/out */
      const opcdata  nodehier,      /*     in */
      opcdata        nodehier_conf  /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
nodehier	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
nodehier_conf	Node hierarchy configuration of type OPCDTYPE_NODEHIER.

Description

Gets the full configuration of the specified node hierarchy.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or nodehier is NULL.
OPC_ERR_INVALID_OUTPARAM	nodehier_conf is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_get_all_layoutgrps()

```
#include opcsvapi.h

int opcnodehier_get_all_layoutgrps (
      opc_connection opc_conn,       /* in/out */
      const opcdata  nodehier,       /*     in */
      opcdata        layoutgrp_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
`layoutgrp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_LAYOUT_GROUP. The returned elements are of type OPCDTYPE_LAYOUT_GROUP.

Description

Returns a list of all layout groups of a node hierarchy. Each node is of type `OPCDTYPE_NODE`.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn`, `nodehier` are NULL or wrong type.
`OPC_ERR_INVALID_OUTPARAM`	`layoutgrp_list` is NULL or wrong type.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.
`OPC_ERR_NODEHIER_NOT_FOUND`	Node hierarchy not found.

OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_get_all_nodes()

```
#include opcsvapi.h

int opcnodehier_get_all_nodes (
      opc_connection opc_conn,  /* in/out */
      const opcdata  nodehier,  /*     in */
      opcdata        node_list  /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
nodehier	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
node_list	Container of type OPCDTYPE_EMPTY or OPCDTYPE_NODE. The returned elements or of type OPCDTYPE_LAYOUT_GROUP.

Description

Returns a list of all nodes of a node hierarchy. Each node is of type OPCDTYPE_NODE. The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The layout group must exist and be specified by either UUID or name and path. The UUID will supersede the name if both are given.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn, nodehier are NULL or wrong type.
OPC_ERR_INVALID_OUTPARAM	node_list is NULL or wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.

OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_get_layoutgrp()

```
#include opcsvapi.h

int opcnodehier_get_layoutgrp (
      opc_connection opc_conn,  /* in/out */
      const opcdata  nodehier,  /*     in */
      opcdata        layoutgrp  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
`layoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.

Description

Gets the full configuration of the specified layout group.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The layout group can be specified by either UUID or name and path. The UUID will supersede the name if both are given. If you want to get the configuration of a nested layout group, you must specify the complete path from the toplevel layout group using the `OPCDATA_PATH` field of the `OPCDTYPE_LAYOUT_GROUP` data structure. Each level must be separated by a slash, for example `level-1/level-2`.

The layout group configuration is checked. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Layout group not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_get_layoutgrps()

```
#include opcsvapi.h

int opcnodehier_get_layoutgrps (
      opc_connection opc_conn,          /* in/out */
      const opcdata  nodehier,          /*     in */
      const opcdata  parent_layoutgrp, /*     in */
      opcdata        layoutgrp_list     /*    out */
      );
```

Parameters

`opc_conn` Connection to the HPOM database.

`nodehier` Node hierarchy configuration of type OPCDTYPE_NODEHIER.

`parentlayoutgrp` Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.

`layoutgrp_list` Container of type OPCDTYPE_EMPTY or OPCDTYPE_LAYOUT_GROUP. The returned elements or of type OPCDTYPE_LAYOUT_GROUP.

Description

Gets a list of all layout groups directly assigned to a layout group. Each layout group is of the type OPCDTYPE_LAYOUT_GROUP.

The node hierarchy must be specified by either UUID or name. If the UUID is given, the name will be ignored.

The layout group can be specified by either UUID or name and path. The UUID will supersede the name if both are given. If you want to query a nested layout group, you must specify the complete path from the toplevel layout group using the `OPCDATA_PATH` field of the `OPCDTYPE_LAYOUT_GROUP` data structure. Each level must be separated by a slash, for example `level-1/level-2`.

If the parent layout group is empty, the top level layout groups of this node hierarchy will be returned.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn, nodehier are NULL or wrong type.
OPC_ERR_INVALID_OUTPARAM	layoutgrp_list is NULL or wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Layout group not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_get_list()

```
#include opcsvapi.h

int opcnodehier_get_list (
      opc_connection opc_conn,      /* in/out */
      opcdata        nodehier_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_NODEHIER. The returned elements are of type OPCDTYPE_NODEHIER.

Description

Gets a list of all node hierarchies with the full configuration of each single node hierarchy.

The parameter `nodehier_list` must be an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_NODEHIER.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or nodehier is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`nodehier_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.
`OPC_ERR_LOCKED_BY_OTHER`	This object is currently locked.
`OPC_ERR_DEADLOCK`	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_get_nodeparent()

```
#include opcsvapi.h

int opcnodehier_get_nodeparent (
      opc_connection opc_conn,     /* in/out */
      const opcdata  nodehier,     /*     in */
      const opcdata  node,         /*     in */
      opcdata        parent_group  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
nodehier	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
node	Node configuration of type OPCDTYPE_NODE.
parent_group	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.

Description

Returns the node layout group that holds the specified node. The node is of the type OPCDTYPE_NODE.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

A node is specified by the node name OPCDATA_NAME and its network type OPCDATA_NETWORK_TYPE, or its IP address OPCDATA_IP_ADDRESS, or its UUID OPCDATA_ID. The node UUID has the highest priority. When the node UUID is set, the name, the network type, and the IP address are ignored.

If the specified node hierarchy or node does not exist, OPC_ERR_NODEHIER_NOT_FOUND is returned.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn, nodehier, node are NULL or wrong type.

OPC_ERR_INVALID_OUTPARAM	parent_group is NULL or wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_NODE_NOT_FOUND	node not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_get_nodes()

```
#include opcsvapi.h

int opcnodehier_get_nodes (
      opc_connection opc_conn,  /* in/out */
      const opcdata  nodehier,  /*     in */
      const opcdata  layoutgrp, /*     in */
      opcdata        node_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
`layoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.
`node_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_NODE. The returned elements or of type OPCDTYPE_LAYOUT_GROUP.

Description

Returns a list of all nodes assigned to the specified layout group. Each node is of the type OPCDTYPE_NODE.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The layout group must exist and must be specified by either the UUID or the name and path. The UUID will supersede the name, if both are given. If you want to query a nested layout group, you must specify the complete path from the toplevel layout group using the `OPCDATA_PATH` field of the `OPCDTYPE_LAYOUT_GROUP` data structure. Each level must be separated by a slash, for example `level-1/level-2`.

The layout group configuration is checked. If the layout group is empty, the toplevel nodes of the node hierarchy will be returned. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn, nodehier, layoutgrp are NULL or wrong type.
OPC_ERR_INVALID_OUTPARAM	node_list is NULL or wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Layout group not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_modify()

```
#include opcsvapi.h

int opcnodehier_modify (
      opc_connection opc_conn,      /* in/out */
      const opcdata  nodehier,      /*     in */
      opcdata        nodehier_conf  /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
nodehier	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
nodehier_conf	Node hierarchy configuration of type OPCDTYPE_NODEHIER.

Description

Modifies the specified node hierarchy. The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The nodehier_conf must contain the full new configuration.

The node hierarchy configuration is checked, before modifying. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition. The name of the node hierarchy must be specified.

If a hierarchy with this name already exists, OPC_ERR_OBJECT_ALREADY_EXISTS is returned and the node hierarchy will not be created.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or nodehier is NULL.
OPC_ERR_INVALID_OUTPARAM	nodehier_conf is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_modify_layoutgrp()

```
#include opcsvapi.h

int opcnodehier_modify_layoutgrp (
      opc_connection opc_conn,       /* in/out */
      const opcdata  nodehier,       /*     in */
      opcdata        layoutgrp,      /*     in */
      opcdata        layoutgrp_conf  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
`layoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.
`layoutgrp_conf`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.

Description

Modifies the specified layout group.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The layout group can be specified by either UUID or name and path. The UUID will supersede the name if both are given. If you want to modify a nested layout group, you must specify the complete path from the toplevel layout group using the `OPCDATA_PATH` field of the `OPCDTYPE_LAYOUT_GROUP` data structure. Each level must be separated by a slash, for example `level-1/level-2`.

The `layoutgrp_conf` must contain the full new configuration.

The layout group configuration is checked. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy or parent not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Layout group not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_move_layoutgrp()

```
#include opcsvapi.h

int opcnodehier_move_layoutgrp (
      opc_connection opc_conn,     /* in/out */
      const opcdata  nodehier,     /*     in */
      const opcdata  to_layoutgrp, /*     in */
      opcdata        layoutgrp     /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
`to_layoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.
`layoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.

Description

Moves the specified layout group into another layout group, called parent.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each, the parent and the layout group itself must exist and be specified be either UUID or name and path. The UUID will supersede the name if both are given. If you want to move a nested layout group, you must specify the complete path from the toplevel layout groups using the `OPCDATA_PATH` field of the `OPCDTYPE_LAYOUT_GROUP` data structure. Each level must be separated by a slash, for example `level-1/level-2`.

The layout group configuration is checked, before moving. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Layout group not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_move_layoutgrps()

```
#include opcsvapi.h

int opcnodehier_move_layoutgrps (
      opc_connection opc_conn,       /* in/out */
      const opcdata  nodehier,       /*     in */
      const opcdata  to_layoutgrp,   /*     in */
      opcdata        layoutgrp_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
`to_layoutgrp`	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.
`layoutgrp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_LAYOUT_GROUP.

Description

Moves the specified layout groups into another layout group, called parent.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each, the parent and the layout groups themselves must exist and be specified be either UUID or name and path. The UUID will supersede the name, if both are given. If you want to move nested layout groups, you must specify the complete path from the toplevel layout groups using the `OPCDATA_PATH` field of the `OPCDTYPE_LAYOUT_GROUP` data structure. Each level must be separated by a slash, for example `level-1/level-2`.

The layout groups configuration is checked, before moving. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	layoutgrp_list is NULL or wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Layout group not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

opcnodehier_move_nodes()

```
#include opcsvapi.h

int opcnodehier_move_nodes (
      opc_connection opc_conn,     /* in/out */
      const opcdata  nodehier,     /*     in */
      const opcdata  to_layoutgrp, /*     in */
      opcdata        node_list     /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
nodehier	Node hierarchy configuration of type OPCDTYPE_NODEHIER.
to_layoutgrp	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP.
node_list	Container of type OPCDTYPE_EMPTY or OPCDTYPE_NODE.

Description

Moves each node in the list to the specified layout group. Each node is of the type OPCDTYPE_NODE. A node is specified by the node name OPCDATA_NAME and its network type OPCDATA_NETWORK_TYPE, or its IP address OPCDATA_IP_ADDRESS, or its UUID OPCDATA_ID. The node UUID has the highest priority. When the node UUID is set, the name, the network type, and the IP address are ignored.

The node hierarchy must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The layout group must exist and be specified by either UUID or name and path. The UUID will supersede the name, if both are given. If you want to move nodes of a nested layout group, you must specify the complete path from the toplevel layout group using the OPCDATA_PATH field of the OPCDTYPE_LAYOUT_GROUP data structure. Each level must be separated by a slash, for example level-1/level-2.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn, nodehier, layoutgrp are NULL or wrong type.
OPC_ERR_INVALID_OUTPARAM	node_list is NULL or wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LAYOUTGROUP_NOT_FOUND	Layout group not found.
OPC_ERR_LOCKED_BY_OTHER	Hierarchy or parent is locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 560

“OPCDTYPE_NODE” on page 574

“OPCDTYPE_NODEHIER” on page 581

“opc_connect()” on page 181

Notification Schedule Configuration API

The Notification Schedule Configuration API provides a set of functions to configure HPOM notification schedules.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see “opcdata_create()” on page 49 or “opcdata_clear()” on page 46) and freeing (see “opcdata_free()” on page 51) the needed memory.

Data Structures

OPCDTYPE_NOTI_SCHEDULE

Usage

To use these functions, it is necessary to connect to the management server as an HPOM user with administrator rights, using the function opc_connect() (see page 181).

1. Use opcdata_create() to create an object of type OPCDTYPE_NOTI_SCHEDULE.
2. Use opcdata_set_str() to set string attributes to, and opcdata_get_str() to get string attributes from the object of type OPCDTYPE_NOTI_SCHEDULE.
3. Use opcdata_set_long() to set and opcdata_get_long() to get integer attributes.
4. Call the appropriate opcnotischedule_* API functions when all necessary attributes are set or retrieved.
5. Use opcdata_free() to free the object of type OPCDTYPE_NOTI_SCHEDULE when you no longer need it.

Prerequisites

The Notification Schedule Configuration API is available only on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

The API can safely be called by multithreaded applications, and it is thread-safe for POSIX Threads. It cannot be safely called in Kernel Threads and it is neither async-cancel safe, async-signal safe, nor fork-safe.

opcnotischedule_add()

```
#include opcsvapi.h

int opcnotischedule_add (
     opc_connection   opc_conn,       /* in/out */
     const char *     service_name,   /* in */
     const opcdata    notischedule    /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
service_name	Notification schedule name.
notischedule	Notification schedule configuration of the type OPCDTYPE_NOTI_SCHEDULE.

Description

Adds a new notification schedule to the HPOM database. The notification schedule must be specified by the name.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Notification schedule already exists.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotischedule_del()” on page 362

“opcnotischedule_get()” on page 364

“opcnotischedule_get_list()” on page 366

“opcnotischedule_modify()” on page 367

opcnotischedule_del()

```
#include opcsvapi.h

int opcnotischedule_del (
      opc_connection  opc_conn,          /* in/out */
      const opcdata   notischedule       /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
notischedule	Notification schedule configuration of the type OPCDTYPE_NOTI_SCHEDULE.

Description

Deletes a notification schedule from the HPOM database. The complete notification schedule must be specified in a data structure of type OPCDTYPE_NOTI_SCHEDULE.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_OBJECT_NOT_FOUND	Notification schedule not found.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotischedule_add()” on page 360

“opcnotischedule_get()” on page 364

“opcnotischedule_get_list()” on page 366

“opcnotischedule_modify()” on page 367

opcnotischedule_get()

```
#include opcsvapi.h

int opcnotischedule_get (
      opc_connection  opc_conn,          /* in/out */
      const char *    service_name,      /* in  */
      const int       day,               /* in  */
      opcdata         notischedules      /* out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`service_name`	Notification schedule name.
`day`	Day must be of type `integer`.
`notischedules`	List of notification schedules (container) of type OPCDTYPE_CONTAINER.

Description

Gets the full configuration of the specified notification schedule from the HPOM database. The notification schedule must be specified by the name or the day, or both. Returns `notischedules`, a list of type OPCDTYPE_CONTAINER.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_OBJECT_NOT_FOUND`	Notification schedule not found.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_CFGUPLD_RUNNING`	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotischedule_add()” on page 360

“opcnotischedule_del()” on page 362

“opcnotischedule_get_list()” on page 366

“opcnotischedule_modify()” on page 367

opcnotischedule_get_list()

```
#include opcsvapi.h

int opcnotischedule_get_list (
      opc_connection  opc_conn,          /* in/out */
      opcdata         notischedules      /* out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
notischedules	Container of type OPCDTYPE_CONTAINER.

Description

Gets a list of all notification schedules stored in the HPOM database.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotischedule_add()” on page 360

“opcnotischedule_del()” on page 362

“opcnotischedule_get()” on page 364

“opcnotischedule_modify()” on page 367

opcnotischedule_modify()

```
#include opcsvapi.h

int opcnotischedule_modify (
      opc_connection  opc_conn,          /* in/out */
      const opcdata   notischedule_old, /* in */
      const opcdata   notischedule_new  /* in */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`notischedule_new`	Notification schedule configuration of type OPCDTYPE_NOTI_SCHEDULE.
`notischedule_old`	Notification schedule configuration of type OPCDTYPE_NOTI_SCHEDULE.

Description

Modifies the notification schedule specified in `notischedule_old`. `notschedule_new` contains the new configuration of the notification schedule.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_OBJECT_NOT_FOUND`	Notification schedule not found.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_CFGUPLD_RUNNING`	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotischedule_add()” on page 360

“opcnotischedule_del()” on page 362

“opcnotischedule_get()” on page 364

“opcnotischedule_get_list()” on page 366

Notification Service Configuration API

The Notification Service Configuration API provides a set of functions to configure HPOM notification services.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see “opcdata_create()” on page 49 or “opcdata_clear()” on page 46) and freeing (see “opcdata_free()” on page 51) the needed memory.

Data Structures

OPCDTYPE_NOTI_SERVICE

Usage

To use these functions, it is necessary to connect to the management server as an HPOM user with administrator rights, using the function opc_connect() (see page 181).

1. Use opcdata_create() to create an object of type OPCDTYPE_NOTI_SERVICE.
2. Use opcdata_set_str() to set string attributes to, and opcdata_get_str() to get string attributes from the object of type OPCDTYPE_NOTI_SERVICE.
3. Use opcdata_set_long() to set and opcdata_get_long() to get integer attributes.
4. Call the appropriate opcnotiservice_* API functions when all necessary attributes are set or retrieved.
5. Use opcdata_free() to free the object of type OPCDTYPE_NOTI_SERVICE when you no longer need it.

Prerequisites

The Notification Service Configuration API is available only on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

The API can safely be called by multithreaded applications, and it is thread-safe for POSIX Threads. It cannot be safely called in Kernel Threads and it is neither async-cancel safe, async-signal safe, nor fork-safe.

opcnotiservice_add()

```
#include opcsvapi.h

int opcnotiservice_add (
     opc_connection   opc_conn,      /* in/out */
     const opcdata    notiservice    /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
notiservice	Notification service configuration of the type OPCDTYPE_NOTI_SERVICE.

Description

Adds a new notification service to the HPOM database. The notification service must be specified by the name.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Notification service already exists.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotiservice_del()” on page 373

“opcnotiservice_get()” on page 374

“opcnotiservice_get_list()” on page 376

“opcnotiservice_modify()” on page 377

opcnotiservice_del()

```
#include opcsvapi.h

int opcnotiservice_del (
      opc_connection  opc_conn,         /* in/out */
      const char *    service_name      /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
service_name	Notification service name.

Description

Deletes a notification service from the HPOM database. The notification service must be specified by the name.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_OBJECT_NOT_FOUND	Notification service not found.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotiservice_add()” on page 371

“opcnotiservice_get()” on page 374

“opcnotiservice_get_list()” on page 376

“opcnotiservice_modify()” on page 377

opcnotiservice_get()

```
#include opcsvapi.h

int opcnotiservice_get (
      opc_connection  opc_conn,         /* in/out */
      const char *    service_name,     /* in  */
      opcdata         notiservice       /* out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
service_name	Notification service name.
notiservice	Notification service configuration of type OPCDTYPE_NOTI_SERVICE.

Description

Gets the full configuration of the specified notification service from the HPOM database. The notification service must be specified by the name.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_OBJECT_NOT_FOUND	Notification service not found.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotiservice_add()” on page 371

“opcnotiservice_del()” on page 373

“opcnotiservice_get_list()” on page 376

“opcnotiservice_modify()” on page 377

opcnotiservice_get_list()

```
#include opcsvapi.h

int opcnotiservice_get_list (
      opc_connection  opc_conn,          /* in/out */
      opcdata         notiservices       /* out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
notiservices	Container of type OPCDTYPE_CONTAINER.

Description

Gets a list of all notification services stored in the HPOM database.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotiservice_add()” on page 371

“opcnotiservice_del()” on page 373

“opcnotiservice_get()” on page 374

“opcnotiservice_modify()” on page 377

opcnotiservice_modify()

```
#include opcsvapi.h

int opcnotiservice_modify (
      opc_connection  opc_conn,         /* in/out */
      const char *    service_name,     /* in */
      const opcdata   notiservice       /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
service_name	Notification service name.
notiservice	Notification service configuration of type OPCDTYPE_NOTI_SERVICE.

Description

Modifies the specified notification service. The notification service must be specified by the name.

notiservice must contain the new configuration.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_OBJECT_NOT_FOUND	Notification service not found.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.30 and later

See Also

“opc_connect()” on page 181

“opcnotiservice_add()” on page 371

“opcnotiservice_del()” on page 373

“opcnotiservice_get()” on page 374

“opcnotiservice_get_list()” on page 376

Policy Configuration API

The Policy Configuration API provides a set of functions to configure policy types, policies and policy groups. Policies are used to configure HPOM agents and represent message sources from which HPOM messages are generated.

General information about policy types, policies and policy groups are contained in opcdata structures of type OPCDTYPE_POLICY_TYPE, OPCDTYPE_POLICY and OPCDTYPE_POLICY_GROUP respectively. Each group has a special API that is used to obtain information about any of its members.

Policies consist of two parts, a header and one or more bodies. Headers contain general policy information like name, type, version, description, and so on, while bodies contain message generation conditions and rules. Separate APIs exist for obtaining header data and downloading policy bodies.

Functions to configure all policy related entities work with their simple representations in an opcdata structure of appropriate type. You can use this information to specify the entity you want to delete, modify or simply obtain data on.

A policy type is identified either by its UUID or its name. A policy is identified by its name, type and version combination, which has to be unique for each policy, or by its UUID. Policy group is identified by its name or UUID. UUID takes precedence over other identifying attributes.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see *opcdata_create(3)* or *opcdata_clear(3)*) and freeing (see *opcdata_free(3)*) the needed memory.

Data Structures

A policy is specified by either the UUID or the name and type. If the UUID is given, a specified name and type will be ignored.

A policy group is specified either by the UUID or the name. If the UUID is given, the specified name will be ignored.

Usage

To use these functions, it is necessary to connect to the management server as administrator or policy administrator using the function `opc_connect`, see page 181.

Prerequisites

This API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

Policy Type API

opcpolicytype_add()

```
#include <opcsvapi.h>

int opcpolicytype_add ( opc_connection  opc_conn,
                        opcdata         policy_type );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policy_type`	Policy of type OPCDTYPE_POLICY_TYPE or OPCDATA_CONTINER

Description

Use the `opcpolicytype_add()` to add a policy type to the HPOM database.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opc(5).

opcpolicytype_add_from_xml()

```
#include <opcsvapi.h>

int opcpolicytype_add_from_xml ( opc_connection  opc_conn,
                                 char *          filename );
```

Parameters

opc_conn	Connection information of the connected user.
filename	Specifies the xml file containing policy type definitions.

Description

Parses the provided XML file and adds the policy types found within to the HPOM database.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3),
opc(5).

opcpolicytype_write_xml

```
#include <opcsvapi.h>

int opcpolicytype_write_xml ( opc_connection  opc_conn,
                              const opcdata   policy_type,
                              const int       policy,
                               const char *    filename );
```

Parameters

opc_conn	Connection information of the connected user.
policy_type	opcdata element of type OPCDTYPE_POLICY_TYPE, identifying a policy type (by name or UUID) for which the XML file is to be written.
policy	Whether policy type policy is to be dumped or not.
filename	XML file containing policy type definitions.

Description

Writes an XML file containing policy type definition for an existing policy type from the HPOM database.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3),
opc(5).

opcpolicytype_delete()

```
#include <opcsvapi.h>

int opcpolicytype_delete ( opc_connection  opc_conn,
                           opcdata         policy_type );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policy_type`	opcdata element of type OPCDTYPE_POLICY_TYPE, identifying a policy type (by name or UUID) to be deleted.

Description

Deletes a policy type from the HPOM database.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3),
opc(5).

opcpolicytype_get()

```
#include <opcsvapi.h>

int opcpolicytype_get ( opc_connection  opc_conn,
                        opcdata         policy_type_in,
                         opcdata         policy_type_out );
```

Parameters

opc_conn	Connection information of the connected user.
policy_type_in	opcdata element of type OPCDTYPE_POLICY_TYPE, identifying a policy type (by name or UUID) to be retrieved.
policy_type_out	opcdata element of type OPCDTYPE_POLICY_TYPE, which will contain full policy type data upon successful invocation of the API.

Description

Retrieves policy type data from the HPOM database.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3),
opc(5).

opcpolicytype_get_template()

```
#include <opcsvapi.h>

int opcpolicytype_get_template ( opc_connection  opc_conn,
                                 opcdata         policy_type_in,
                                  char **          filename );
```

Parameters

opc_conn	Connection information of the connected user.
policy_type_in	opcdata element of type OPCDTYPE_POLICY_TYPE, identifying a policy type (by name or UUID) for which the policy is requested.
filename	Contains the filename to which the policy type policy should be written to.

Description

The function opcpolicytype_get_template() is used for backward compatibility only. This function dumps the policy type policy to the file system.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3),
opc(5).

opcpolicytype_modify()

```
#include <opcsvapi.h>

int opcpolicytype_modify ( opc_connection  opc_conn,
                           opcdata         policy_type_old,
                            opcdata         policy_type_new );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policy_type_old`	opcdata element of type OPCDTYPE_POLICY_TYPE, identifying a policy type (by name or UUID) to be modified.
`policy_type_new`	opcdata element of type OPCDTYPE_POLICY_TYPE, containing values that should be modified.

Description

Modifies the editor or the policy type policy of an existing policy type.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3),
opc(5).

opcpolicytype_get_name_by_uuid()

```
#include <opcsvapi.h>

int opcpolicytype_get_name_by_uuid ( opc_connection  opc_conn,
                        const char *     id,
                        char **          name_used_by_editor,
                         char **         name_used_on_agent );
```

Parameters

`opc_conn` Connection information of the connected user.

`id` UUID of the policy type to be retrieved.

`name_used_by_editor` Name of the policy type as known on the HPOM management server.

`name_used_on_agent` Name of the policy type as known on the HPOM agent side.

Description

Returns the names of the policy type as known on server and agent.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3),
opc(5).

opcpolicytype_get_uuid_by_name()

```
#include <opcsvapi.h>

int opcpolicytype_get_uuid_by_name ( opc_connection  opc_conn,
                                     const char *    name,
                                     char *          id );
```

Parameters

opc_conn	Connection information of the connected user.
name	Name of the policy type, as known on the server.
id	Contains the UUID of the policy type upon successful invocation of the API.

Description

Returns the policy type UUID.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3),
opc(5).

Policy Configuration API

opcpolicy_add()

```
#include <opcsvapi.h>

int opcpolicy_add ( opc_connection  opc_conn,
                    char *          filename,
                    int             check,
                    int             upload_mode,
                     opcdata *       policies );
```

Parameters

opc_conn	Connection information of the connected user.
filename	Full path to the policy header or a directory containing policy headers.
check	Determines if the check callback will be run on the policy bodies prior to upload to database.
upload_mode	Modifies the behavior of the upload if the policy already exists in the HPOM database.
policies	Output parameter that contains the opcdata elements of the uploaded policies.

Description

Uploads a policy or policies to the HPOM database from the provided policy header, or the directory containing policy headers.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_get()

```
#include <opcsvapi.h>

int opcpolicy_get ( opc_connection  opc_conn,
                    const opcdata   policy,
                    int             readable,
                    int             deployment,
                     char *          dirname );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policy`	opcdata element of type OPCDTYPE_POLICY, containing data that uniquely identifies a policy.
`readable`	Policy header naming.
`deployment`	Policy naming control.
`dirname`	Full path to the directory into which the policy is to be dumped.

Description

Dumps the policy header and bodies to the file system.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_get_data()

```
#include <opcsvapi.h>

int opcpolicy_get_data ( opc_connection  opc_conn,
                         const opcdata   policy,
                          opcdata         policy_data );
```

Parameters

opc_conn	Connection information of the connected user.
policy	opcdata element of type OPCDTYPE_POLICY, containing data identifying a policy or policies.
policy_data	opcdata element of type OPCDTYPE_CONTAINER.

Description

Retrieves policy data from the database.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_modify()

```
#include <opcsvapi.h>

int opcpolicy_modify ( opc_connection  opc_conn,
                       const opcdata   policy,
                        const opcdata   new_policy );
```

Parameters

opc_conn	Connection information of the connected user.
policy	opcdata element of type OPCDTYPE_POLICY, containing data identifying a policy or policies.
new_policy	opcdata element of type OPCDTYPE_POLICY, containing data to be modified.

Description

Modifies policy data.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_delete()

```
#include <opcsvapi.h>

int opcpolicy_delete ( opc_connection  opc_conn,
                       const opcdata   policy );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policy`	opcdata element of type OPCDTYPE_POLICY, containing data identifying a policy or policies.

Description

Deletes policy from the HPOM database.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_copy()

```
#include <opcsvapi.h>

int opcpolicy_copy ( opc_connection  opc_conn,
                     const opcdata   policy,
                     char *          new_name,
                      char *          new_version );
```

Parameters

opc_conn	Connection information of the connected user.
policy	opcdata element of type OPCDTYPE_POLICY, containing data identifying a policy or a container
new_name	Name of the new policy or container.
new_version	Version of the newly created policy. Argument has no relevance, if policy identifies a container.

Description

Copies a policy or a policy container to a new one.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_edit()

```
#include <opcsvapi.h>

int opcpolicy_edit ( opc_connection  opc_conn,
                     opcdata         policy,
                     long            body_number,
                     char *          dirname,
                      char **         edit_string );
```

Parameters

opc_conn	Connection information of the connected user.
policy	opcdata element of type OPCDTYPE_POLICY, containing data identifying a policy.
body_number	Ordinal number of the policy body, starting from 1.
dirname	Directory where the policy body is to be dumped.
edit_string	Contains the string that can be used to edit the policy body.

Description

Dumps policy body, runs edit callback and produces the editor invocation string.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_edit_body()

```
#include <opcsvapi.h>

int opcpolicy_edit_body ( opc_connection  opc_conn,
                          opcdata         policy,
                          char *          suffix,
                          char *          dirname,
                           char **         edit_string );
```

Parameters

opc_conn	Connection information of the connected user.
policy	opcdata element of type OPCDTYPE_POLICY, containing data identifying a single policy.
suffix	Suffix of the policy body.
dirname	Directory where the policy body is to be dumped.
edit_string	Contains the string that can be used to edit the policy body.

Description

Identical to opcpolicy_edit(), except that the body is referenced by the suffix and not the ordinal number.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_get_list()

```
#include <opcsvapi.h>

int opcpolicy_get_list ( opc_connection  opc_conn,
                         opcdata         policies );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policies`	opcdata element of type OPCDTYPE_CONTAINER, contains elements of OPCDTYPE_POLICY type, each representing one policy that exists in the HPOM database.

Description

Retrieves the list of all policies present in the HPOM database.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_get_list_by_type()

```
#include <opcsvapi.h>

int opcpolicy_get_list_by_type ( opc_connection  opc_conn,
                                 opcdata         poltype,
                                  opcdata         policies );
```

Parameters

opc_conn	Connection information of the connected user.
poltype	opcdata element of type OPCDTYPE_POLICY_TYPE, containing data identifying a single policy type.
policies	opcdata element of type OPCDTYPE_CONTAINER, contains elements of OPCDTYPE_POLICY type, each representing one policy of type defined in poltype that exists in the HPOM database.

Description

Retrieves policy data from the database.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicy_header_create()

```
#include <opcsvapi.h>

int opcpolicy_header_create ( opc_connection  opc_conn,
                              opcdata         policy,
                              int             readable,
                               char *          dirname );
```

Parameters

opc_conn	Connection information of the connected user.
policy	opcdata element of type OPCDTYPE_POLICY, containing data identifying a single policy or containing enough data to prepare a new policy header.
readable	Defines the naming of the policy header.
dirname	Directory where the policy body is dumped. This is a mandatory argument, if set to NULL, API will return an error.

Description

Dumps the policy header of an existing policy, or creates an empty one.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_DB_INCONSISTENT	The database is inconsistent.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicybody_get()

```
#include <opcsvapi.h>

int opcpolicybody_get ( opc_connection  opc_conn,
                        opcdata         policy,
                        long            body_number,
                        char **         encoding,
                         char *          dirname );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policy`	opcdata element of type OPCDTYPE_POLICY, containing data identifying a single policy.
`body_number`	Ordinal number of the policy body, starting from 1. If body_number is set to 0, all policy bodies will be dumped.
`encoding`	Contains the encoding of the dumped policy body. If multiple bodies are dumped, the value of encoding is undefined.
`dirname`	Directory where the policy body is dumped. If dirname is NULL, `/var/opt/OV/share/tmp/OpC/mgmt_sv/policyedit` is used.

Description

Dumps the requested policy body to the file system.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicybody_modify()

```
#include <opcsvapi.h>

int opcpolicybody_modify ( opc_connection  opc_conn,
                           opcdata         policy,
                           long            body_number,
                           int             check,
                           char **         encoding,
                            char *          filename );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policy`	opcdata element of type OPCDTYPE_POLICY, containing data identifying a single policy.
`body_number`	Ordinal number of the policy body, starting from 1. If body_number is set to 0, all policy bodies will be replaced.
`check`	Determines if the check callback will be run on the policy bodies prior to upload to database.
`encoding`	Contains the encoding of the replaced policy body. If not set, the existing one is kept.
`filename`	Full path to the file to replace the policy body.

Description

Replaces the policy body with the contents of the provided file.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

opcpolicybody_modify_by_name()

```
#include <opcsvapi.h>

int opcpolicybody_modify_by_name ( opc_connection  opc_conn,
                                   opcdata         policy,
                                   char *          suffix,
                                   int             check,
                                   char **         encoding,
                                    char *         filename );
```

Parameters

`opc_conn`	Connection information of the connected user.
`policy`	opcdata element of type OPCDTYPE_POLICY, containing data identifying a single policy.
`body_number`	Ordinal number of the policy body, starting from 1. If body_number is set to 0, all policy bodies will be replaced.
`suffix`	Suffix of the policy body.
`check`	Determines if the check callback will be run on the policy bodies prior to upload to database.
`encoding`	Contains the encoding of the replaced policy body. If not set, the existing one is kept.
`filename`	Full path to the file to replace the policy body.

Description

Identical to opcpolicybody_modify(), except that the policy body is referenced by suffix instead of its ordinal number.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`policy` is NULL. or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_DB_INCONSISTENT`	The database is inconsistent.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

HPOM 9.00 and later.

See Also

opc_connect(3),
opc_disconnect(3),
opcdata_api(3), opciter_api(3), opcpolicytype_api(3)
opc(5).

Trouble Ticket Configuration API

The Trouble Ticket Configuration API provides a set of functions to configure HPOM for trouble ticket systems.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see “opcdata_create()” on page 49 or “opcdata_clear()” on page 46) and freeing (see “opcdata_free()” on page 51) the needed memory.

Usage

To use these functions, it is necessary to connect to the management server as an HPOM user with administrator rights, using the function opc_connect() (see page 181).

Prerequisites

The Trouble Ticket Configuration API is available only on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opctroubleticket_get()

```
#include opcsvapi.h

int opctroubleticket_get (
      const opc_connection  opc_conn, /* in/out */
      char **               tt_call,  /* out */
      unsigned char *       tt_active /* out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
tt_call	Command call to the trouble ticket system.
tt_active	Trouble ticket mode: enabled or disabled.

Description

Retrieves information about HPOM configuration for trouble ticket systems from the HPOM database.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.27 and later

See Also

“opc_connect()” on page 181

“opctroubleticket_set()” on page 422

opctroubleticket_set()

```
#include opcsvapi.h

int opctroubleticket_set (
      const opc_connection  opc_conn, /* in/out */
      char **               tt_call,  /* in */
      unsigned char *       tt_active /* in */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
tt_call	Command call to the trouble ticket system.
tt_active	Trouble ticket mode: enabled or disabled.

Description

Configures HPOM to forward messages to a trouble ticket system and stores the configuration in the HPOM database.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_CFGUPLD_RUNNING	Configuration upload in progress.

Versions

8.27 and later

See Also

“opc_connect()” on page 181

“opcappl_add()” on page 190

User Profile Configuration API

The user profile API provides a set of functions to configure HPOM user profiles.

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see *opcdata_create(3)* or *opcdata_clear(3)*) and freeing (see *opcdata_free(3)*) the needed memory.

Data Structures

OPCDTYPE_USER_CONFIG

A profile is specified either by name or by the UUID. If the UUID is given, a specified name will be ignored.

Usage

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect(), see *opc_connect(3)*.

Prerequisites

The User Profile Configuration API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcprofile_add()

```
#include opcsvapi.h

 opcprofile_add (
      opc_connection opc_conn, /* in/out */
      opcdata        profile   /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
profile	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Adds a new HPOM profile. Only the attributes of the profile will be set, not assignments of other profiles, applications, responsibilities or node hierarchies.

The profile configuration is checked before creation. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

The name of the profile must be specified. If a profile with this name already exists, OPC_ERR_OBJECT_ALREADY_EXISTS is returned and the profile will not be created.

The UUID of the created object will be returned in the opcdata structure if successful.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or profile is NULL or profile has invalid role.
OPC_ERR_INVALID_OUTPARAM	profile is NULL or invalid type.
OPC_ERR_INVALID_NAME	The name contains invalid characters or is empty.
OPC_ERR_ACCESS_DENIED	profile has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Profile already exists.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

“opcdata_create()” on page 49

opcprofile_assign_applgrps()

```
#include opcsvapi.h

 opcprofile_assign_applgrps (
      opc_connection opc_conn,      /* in/out */
      const opcdata  profile,       /*     in */
      opcdata        applgrp_list   /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
profile	User configuration of type OPCDTYPE_USER_CONFIG.
applgrp_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Assigns application groups to a specified HPOM profile.

profile must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each application group in the container can be specified by either the name or the UUID.

If an application group could not be assigned to profile, OPC_ERR_NOT_COMPLETELY_DONE is returned and the specific error code is in the OPCDATA_STATUS field of the opcdata structure of the application group.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or profile is NULL or profile has invalid role.
OPC_ERR_INVALID_OUTPARAM	applgrp_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all application groups could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Application group already assigned to that profile

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_assign_appls()

```
#include opcsvapi.h

 opcprofile_assign_appls (
      opc_connection opc_conn,  /* in/out */
      const opcdata  profile,   /*     in */
      opcdata        appl_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`appl_list`	Container of type OPCDTYPE_APPL_CONFIG.

Description

Assigns applications to a specified HPOM `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each application in the container can be specified by either the name or the UUID.

If an application could not be assigned to `profile`, `OPC_ERR_NOT_COMPLETELY_DONE` is returned and the specific error code is in the `OPCDATA_STATUS` field of the opcdata structure of the application.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`appl_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all applications could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Application already assigned to that profile.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_assign_profiles()

```
#include opcsvapi.h

 opcprofile_assign_profiles (
      opc_connection opc_conn,      /* in/out */
      const opcdata  profile,       /*     in */
      opcdata        profile_list   /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
profile	User configuration of type OPCDTYPE_USER_CONFIG.
profile_list	Container of type OPCDTYPE_USER_CONFIG.

Description

Assigns profiles to a specified HPOM profile.

Each profile must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a profile could not be assigned to profile, OPC_ERR_NOT_COMPLETELY_DONE is returned and the corresponding error code in the OPCDATA_STATUS field for the profile is set.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or profile is NULL or profile has invalid role.
OPC_ERR_INVALID_OUTPARAM	profile_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.

OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all profiles could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	`profile` already assigned to that `profile`.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_assign_resps()

```
#include opcsvapi.h

 opcprofile_assign_resps (
      opc_connection opc_conn,  /* in/out */
      const opcdata  profile,   /*     in */
      opcdata        resp_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`resp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_RESP_ENTRY.

Description

Assigns the defined responsibilities to a specified HPOM `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each responsibility must be specified by message group name and a node group. The node group can be specified either by name or the UUID.

The return status for each assignment is set for the `resp_list` element. If an error occurs, the remaining assignments will be processed and `OPC_ERROR_NOT_COMPLETELY_DONE` will be returned.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`resp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all responsibilities could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Responsibility already assigned to that profile.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“OPCDTYPE_USER_RESP_ENTRY” on page 587

“opc_connect()” on page 181

opcprofile_deassign_applgrps()

```
#include opcsvapi.h

 opcprofile_deassign_applgrps (
      opc_connection opc_conn,      /* in/out */
      const opcdata  profile,       /*     in */
      opcdata        applgrp_list   /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
profile	User configuration of type OPCDTYPE_USER_CONFIG.
applgrp_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Deassigns application groups from a specified HPOM profile.

profile must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each application group in the container can be specified by either the name or the UUID.

If an application group could not be deassigned from the profile, OPC_ERR_NOT_COMPLETELY_DONE is returned and the specific error code is in the OPCDATA_STATUS field of the opcdata structure of the application group.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or profile is NULL or profile has invalid role.
OPC_ERR_INVALID_OUTPARAM	applgrp_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all application groups could be deassigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Application group is not assigned to that profile.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_deassign_appls()

```
#include opcsvapi.h

 opcprofile_deassign_appls (
      opc_connection opc_conn,  /* in/out */
      const opcdata  profile,   /*     in */
      opcdata        appl_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`appl_list`	Container of type OPCDTYPE_APPL_GROUP.

Description

Deassigns applications from a specified HPOM `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each application in the container can be specified by either the name or the UUID.

If an application could not be deassigned from the `profile`, `OPC_ERR_NOT_COMPLETELY_DONE` is returned and the specific error code is in the `OPCDATA_STATUS` field of the opcdata structure of the application.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`appl_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all applications could be deassigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Application is not assigned to that profile

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_deassign_profiles()

```
#include opcsvapi.h

 opcprofile_deassign_profiles (
      opc_connection opc_conn,      /* in/out */
      const opcdata  profile,       /*     in */
      opcdata        profile_list   /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
profile	User configuration of type OPCDTYPE_USER_CONFIG.
profile_list	Container of type OPCDTYPE_USER_CONFIG.

Description

Deassigns profiles from a specified HPOM profile.

profile must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each profile in profile_list must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If profiles could not be deassigned from profile, OPC_ERR_NOT_COMPLETELY_DONE is returned and the corresponding error code in the OPCDATA_STATUS field for the profile is set.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or profile is NULL or profile has invalid role.
OPC_ERR_INVALID_OUTPARAM	profile_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all profiles could be deassigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Profile is not assigned to that profile.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_deassign_resps()

```
#include opcsvapi.h

 opcprofile_deassign_resps (
      opc_connection opc_conn,  /* in/out */
      const opcdata  profile,   /*     in */
      opcdata        resp_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`resp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_RESP_ENTRY.

Description

Deassigns the defined responsibilities from a specified HPOM `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each responsibility must be specified by message group name and a node group. The node group can be specified either by name or the UUID.

The return status for each assignment is set for the `resp_list` element. If an error occurs it will be tried to processes the remaining assignments and `OPC_ERROR_NOT_COMPLETELY_DONE` will be returned.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`resp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all responsibilities could be deassigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Responsibility not assigned for that profile.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“OPCDTYPE_USER_RESP_ENTRY” on page 587

“opc_connect()” on page 181

opcprofile_delete()

```
#include opcsvapi.h

 opcprofile_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        profile   /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Deletes the specified `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The `profile` will be deassigned from all users.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.
`OPC_ERR_PROFILE_NOT_FOUND`	Profile not found.
`OPC_ERR_LOCKED_BY_OTHER`	This object is currently locked.
`OPC_ERR_DEADLOCK`	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_get()

```
#include opcsvapi.h

 opcprofile_get (
      opc_connection opc_conn,     /* in/out */
      const opcdata  profile,      /*     in */
      opcdata        profile_conf  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`profile_conf`	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Gets the full configuration of the specified `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`profile_conf` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.
`OPC_ERR_PROFILE_NOT_FOUND`	Profile not found.

OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_get_applgrps()

```
#include opcsvapi.h

 opcprofile_get_applgrps (
      opc_connection opc_conn,      /* in/out */
      opcdata        profile,       /*     in */
      opcdata        applgrp_list   /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`applgrp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_APPL_GROUP. The returned elements are of type OPCDTYPE_APPL_GROUP.

Description

Gets a list of all direct assigned applications of the specified `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The configuration of each application is returned in the container.

The parameter `applgrp_list` must be an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_APPL_GROUP.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`applgrp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_get_appls()

```
#include opcsvapi.h

 opcprofile_get_appls (
      opc_connection opc_conn,  /* in/out */
      opcdata        profile,   /*     in */
      opcdata        appl_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`appl_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_APPL_CONFIG. The returned elements are of type OPCDTYPE_APPL_CONFIG.

Description

Gets a list of all directly assigned applications of the specified `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The configuration of each application is returned in the container.

The parameter `appl_list` must be an OPCDTYPE_CONTAINER of the type OPCDTYPE_EMPTY or OPCDTYPE_APPL_CONFIG.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`appl_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_get_list()

```
#include opcsvapi.h

 opcprofile_get_list (
      opc_connection opc_conn,     /* in/out */
      opcdata        profile_list  /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
profile_list	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_CONFIG. The returned elements are of type OPCDTYPE_USER_CONFIG.

Description

Gets a list of all known HPOM profiles.

The parameter profile_list is an OPCDTYPE_CONTAINER of the type OPCDTYPE_EMPTY or OPCDTYPE_USER_CONFIG.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or profile_list is NULL.
OPC_ERR_INVALID_OUTPARAM	profile_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_get_profiles()

```
#include opcsvapi.h

 opcprofile_get_profiles (
      opc_connection opc_conn,     /* in/out */
      opcdata        profile,      /*     in */
      opcdata        profile_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`profile_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_CONFIG. The returned elements are of type OPCDTYPE_USER_CONFIG.

Description

Gets a list of all assigned profiles for an HPOM `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The parameter `profile_list` is an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_USER_CONFIG.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile_list` is NULL, or invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`profile_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.

OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcprofile_get_resps()

```
#include opcsvapi.h

 opcprofile_get_resps (
      opc_connection opc_conn,  /* in/out */
      opcdata        profile,   /*     in */
      opcdata        resp_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`resp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_RESP_ENTRY. The returned are of type OPCDTYPE_USER_RESP_ENTRY.

Description

Gets a list of all assigned responsibilities of a specified HPOM `profile`.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The parameter `resp_list` is an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_USER_RESP_ENTRY.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`resp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.

OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“OPCDTYPE_USER_RESP_ENTRY” on page 587

“opc_connect()” on page 181

opcprofile_modify()

```
#include opcsvapi.h

 opcprofile_modify (
      opc_connection opc_conn,     /* in/out */
      const opcdata  profile,      /*     in */
      opcdata        mod_profile   /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`profile`	User configuration of type OPCDTYPE_USER_CONFIG.
`mod_profile`	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Modifies the attributes of HPOM profiles.

`profile` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

`mod_profile` must contain the full new configuration.

The profile configuration is checked before modification. If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition. The name of the node hierarchy must be specified.

If a profile with this name already exists, `OPC_ERR_OBJECT_ALREADY_EXISTS` is returned and the profile will not be created.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile` is NULL or `mod_profile` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`mod_profile` is NULL or invalid.

OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_PROFILE_NOT_FOUND	Profile not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_OBJECT_ALREADY_EXISTS	Name of mod_profile already exists.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

User Configuration API

The user API provides a set of functions to configure HPOM users (an administrator, operators and policy administrators).

Error information is written to the `/var/opt/OV/log/System.txt` and `/var/opt/OV/log/System.bin` files on the management server.

Memory for the configuration data is allocated on the heap. The caller is responsible for allocating (see *opcdata_create(3)* or *opcdata_clear(3)*) and freeing (see *opcdata_free(3)*) the needed memory.

Data Structures

`OPCDTYPE_USER_CONFIG`

A user is specified either by name or by the UUID. If the UUID is given, a specified name will be ignored.

Usage

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect(), see *opc_connect(3)*.

Prerequisites

The User Configuration API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opcuser_add()

```
#include opcsvapi.h

 opcuser_add (
      opc_connection opc_conn, /* in/out */
      opcdata        user      /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
user	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Adds a new HPOM user (operator or policy administrator). Only the attributes of the user will be set, not assignments of profiles, applications, responsibilities or node hierarchy.

The user configuration is checked before creation. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition. The name of the user must be specified.

If a user with this name already exists, OPC_ERR_OBJECT_ALREADY_EXISTS is returned and the user will not be created.

The UUID of the created object will be returned in the opcdata structure if successful.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn is NULL or user has invalid role.
OPC_ERR_INVALID_OUTPARAM	user is NULL or invalid type.
OPC_ERR_INVALID_NAME	The name contains invalid characters or is empty.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	User name already exists.
OPC_WARN_PAM_ENABLED	User added successfully but password may be ineffective because PAM is enabled.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

“opcdata_create()” on page 49

opcuser_assign_applgrps()

```
#include opcsvapi.h

 opcuser_assign_applgrps (
      opc_connection opc_conn,     /* in/out */
      const opcdata  user,         /*     in */
      opcdata        applgrp_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`applgrp_list`	Container of type OPCDTYPE_APPL_GROUP.

Description

Assigns application groups to a specified HPOM user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each application group in the container can be specified by either the name or the UUID.

If an application group could not be assigned to `user`, `OPC_ERR_NOT_COMPLETELY_DONE` is returned and the specific error code can be found in the `OPCDATA_STATUS` field of the opcdata structure of the application group.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`applgrp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all application groups could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Application group already assigned to user.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_assign_appls()

```
#include opcsvapi.h

 opcuser_assign_appls (
      opc_connection opc_conn,  /* in/out */
      const opcdata  user,      /*     in */
      opcdata        appl_list  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
user	User configuration of type OPCDTYPE_USER_CONFIG.
appl_list	Container of type OPCDTYPE_APPL_CONFIG.

Description

Assigns applications to a specified HPOM user.

user must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each application in the container can be specified by either the name or the UUID.

If an application could not be assigned to user, OPC_ERR_NOT_COMPLETELY_DONE is returned and the specific error code can be found in the OPCDATA_STATUS field of the opcdata structure of the application.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or user is NULL or user has invalid role.
OPC_ERR_INVALID_OUTPARAM	appl_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all applications could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Application already assigned to user.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_assign_nodehier()

```
#include opcsvapi.h

 opcuser_assign_nodehier (
      opc_connection opc_conn, /* in/out */
      const opcdata  user,     /*     in */
      opcdata        nodehier  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.

Description

Assigns a node hierarchy to a specified HPOM user.

`user` must be specified either by name or the UUID. If the UUID is given, the name will be ignored.

The node hierarchy must be specified either by name or the UUID. If the UUID is given, the name will be ignored.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`nodehier` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_NODEHIER_NOT_FOUND	Node hierarchy not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 581

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_assign_profiles()

```
#include opcsvapi.h

 opcuser_assign_profiles (
      opc_connection opc_conn,     /* in/out */
      const opcdata  user,         /*     in */
      opcdata        profile_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`profile_list`	Container of type OPCDTYPE_USER_CONFIG.

Description

Assigns profiles to a specified HPOM user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each profile must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a profile could not be assigned to `user`, `OPC_ERR_NOT_COMPLETELY_DONE` is returned and the corresponding error code in the `OPCDATA_STATUS` field for the profile is set.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`profile_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all profiles could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Profile already assigned to that user.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_assign_resps()

```
#include opcsvapi.h

 opcuser_assign_resps (
      opc_connection opc_conn,  /* in/out */
      const opcdata  user,      /*     in */
      opcdata        resp_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`resp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_RESP_ENTRY.

Description

Assigns the defined responsibilities to a specified HPOM user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each responsibility must be specified by message group name and a node group. The node group can be specified either by name or the UUID.

The return status for each assignment is set for the `resp_list` element. If an error occurs the remaining assignments will be processed and `OPC_ERROR_NOT_COMPLETELY_DONE` will be returned.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`resp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all responsibilities could be assigned.
OPC_ERR_OBJECT_ALREADY_ASSIGNED	Responsibility already assigned to that user.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“OPCDTYPE_USER_RESP_ENTRY” on page 587

“opc_connect()” on page 181

opcuser_deassign_applgrps()

```
#include opcsvapi.h

 opcuser_deassign_applgrps (
      opc_connection opc_conn,     /* in/out */
      const opcdata  user,         /*     in */
      opcdata        applgrp_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`applgrp_list`	Container of type OPCDTYPE_APPL_GROUP.

Description

Deassigns application groups from a specified HPOM user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each application group in the container can be specified by either the name or the UUID.

If an application group could not be deassigned from `user`, `OPC_ERR_NOT_COMPLETELY_DONE` is returned and the specific error code can be found in the `OPCDATA_STATUS` field of the opcdata structure of the application group.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`applgrp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all application groups could be deassigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Application group is not assigned to user.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_deassign_appls()

```
#include opcsvapi.h

 opcuser_deassign_appls (
      opc_connection opc_conn,  /* in/out */
      const opcdata  user,      /*     in */
      opcdata        appl_list  /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
user	User configuration of type OPCDTYPE_USER_CONFIG.
appl_list	Container of type OPCDTYPE_APPL_CONFIG.

Description

Deassigns applications from a specified HPOM user.

user must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each application in the container can be specified by either the name or the UUID.

If an application could not be deassigned from user, OPC_ERR_NOT_COMPLETELY_DONE is returned and the specific error code can be found in the OPCDATA_STATUS field of the opcdata structure of the application.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or user is NULL or user has invalid role.
OPC_ERR_INVALID_OUTPARAM	appl_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all applications could be deassigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Application is not assigned to user.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_deassign_profiles()

```
#include opcsvapi.h

 opcuser_deassign_profiles (
      opc_connection opc_conn,     /* in/out */
      const opcdata  user,         /*     in */
      opcdata        profile_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`profile_list`	Container of type OPCDTYPE_USER_RESP_ENTRY.

Description

Deassigns profiles from a specified HPOM user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each profile must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a profile could not be deassigned from `user`, `OPC_ERR_NOT_COMPLETELY_DONE` is returned and the corresponding error code in the `OPCDATA_STATUS` field for the profile is set.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`profile_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all profiles could be deassigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Profile is not assigned to `user`.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_deassign_resps()

```
#include opcsvapi.h

 opcuser_deassign_resps (
      opc_connection opc_conn,  /* in/out */
      const opcdata  user,      /*     in */
      opcdata        resp_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`resp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_RESP_ENTRY.

Description

Deassigns the defined responsibilities from a specified HPOM user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

Each responsibility must be specified by message group name and a node group. The node group can be specified either by name or the UUID.

The return status for each deassignment is set for the `resp_list` element. If an error occurs the remaining deassignments will be processed and `OPC_ERROR_NOT_COMPLETELY_DONE` will be returned.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`resp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_NOT_COMPLETELY_DONE	Not all responsibilities could be deassigned.
OPC_ERR_OBJECT_NOT_ASSIGNED	Responsibility not assigned to user.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“OPCDTYPE_USER_RESP_ENTRY” on page 587

“opc_connect()” on page 181

opcuser_delete()

```
#include opcsvapi.h

 opcuser_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        user      /* in/out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
user	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Deletes the specified user.

user must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or user is NULL or user has invalid role.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_get()

```
#include opcsvapi.h

 opcuser_get (
      opc_connection opc_conn,  /* in/out */
      const opcdata  user,      /*     in */
      opcdata        user_conf  /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
user	User configuration of type OPCDTYPE_USER_CONFIG.
user_conf	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Gets the full configuration of the specified user.

user must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or user is NULL or user has invalid role.
OPC_ERR_INVALID_OUTPARAM	user_conf is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.

OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_get_applgrps()

```
#include opcsvapi.h

 opcuser_get_applgrps (
      opc_connection opc_conn,     /* in/out */
      opcdata        user,         /*     in */
      opcdata        applgrp_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`applgrp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_APPL_GROUP. The returned elements are of type OPCDTYPE_APPL_GROUP.

Description

Gets a list of all directly assigned applications of the specified user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The configuration of each application is returned in the container.

The parameter `applgrp_list` is an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_APPL_GROUP.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`applgrp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 551

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_get_appls()

```
#include opcsvapi.h

 opcuser_get_appls (
      opc_connection opc_conn,  /* in/out */
      opcdata        user,      /*     in */
      opcdata        appl_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`appl_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_APPL_CONFIG. The returned elements are of type OPCDTYPE_APPL_CONFIG.

Description

Gets a list of all directly assigned applications of the specified user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The configuration of each application is returned in the container.

The parameter `appl_list` must be an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_APPL_CONFIG.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`appl_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 548

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_get_list()

```
#include opcsvapi.h

 opcuser_get_list (
      opc_connection opc_conn,  /* in/out */
      opcdata        user_list  /*    out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_CONFIG. The returned elements are of type OPCDTYPE_USER_CONFIG.

Description

Gets a list of all known HPOM users.

The parameter `user_list` must be an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_USER_CONFIG.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user_list` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`user_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.
`OPC_ERR_LOCKED_BY_OTHER`	This object is currently locked.
`OPC_ERR_DEADLOCK`	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_get_nodehier()

```
#include opcsvapi.h

 opcuser_get_nodehier (
      opc_connection opc_conn, /* in/out */
      const opcdata  user,     /*     in */
      opcdata        nodehier  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`nodehier`	Node hierarchy configuration of type OPCDTYPE_NODEHIER.

Description

Gets the assigned node hierarchy for a specified HPOM user.

`user` must be specified either by name or the UUID.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `user` is NULL or `user` has invalid role.
`OPC_ERR_INVALID_OUTPARAM`	`nodehier` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.
`OPC_ERR_USER_NOT_FOUND`	User not found.
`OPC_ERR_LOCKED_BY_OTHER`	This object is currently locked.

OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 581

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_get_profiles()

```
#include opcsvapi.h

 opcuser_get_profiles (
      opc_connection opc_conn,     /* in/out */
      opcdata        user,         /*     in */
      opcdata        profile_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`profile_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_CONFIG. The returned elements are of type OPCDTYPE_USER_CONFIG.

Description

Gets a list of all assigned profiles for an HPOM user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The parameter `profile_list` must be an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_USER_CONFIG.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `profile_list` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`profile_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.

OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

opcuser_get_resps()

```
#include opcsvapi.h

 opcuser_get_resps (
      opc_connection opc_conn,  /* in/out */
      opcdata        user,      /*     in */
      opcdata        resp_list  /* in/out */
      );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`user`	User configuration of type OPCDTYPE_USER_CONFIG.
`resp_list`	Container of type OPCDTYPE_EMPTY or OPCDTYPE_USER_RESP_ENTRY. The returned are of type OPCDTYPE_USER_RESP_ENTRY.

Description

Gets a list of all assigned responsibilities of a specified HPOM user.

`user` must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

The parameter `resp_list` must be an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_USER_RESP_ENTRY.

If a field contains an improper value, the function returns a positive error value corresponding to the `OPCDATA_*` definition.

Return Values

`OPC_ERR_OK`	Access successful.
`OPC_ERR_INVALID_INPARAM`	`opc_conn` or `resp_list` is NULL.
`OPC_ERR_INVALID_OUTPARAM`	`resp_list` is NULL or invalid.
`OPC_ERR_ACCESS_DENIED`	User has no administrator rights.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.

Versions

5.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 542

“OPCDTYPE_USER_CONFIG” on page 586

“OPCDTYPE_USER_RESP_ENTRY” on page 587

“opc_connect()” on page 181

opcuser_modify()

```
#include opcsvapi.h

 opcuser_modify (
      opc_connection opc_conn, /* in/out */
      const opcdata  user,     /*     in */
      opcdata        mod_user  /*    out */
      );
```

Parameters

opc_conn	Connection to the HPOM database.
user	User configuration of type OPCDTYPE_USER_CONFIG.
mod_user	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Modifies the attributes of HPOM users.

user must be specified by either the UUID or the name. If the UUID is given, the name will be ignored.

mod_user must contain the full new configuration.

The user configuration is checked before modification. If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

If a user with the specified name already exists, OPC_ERR_OBJECT_ALREADY_EXISTS will be returned. The name of the node hierarchy must be specified.

If a field contains an improper value, the function returns a positive error value corresponding to the OPCDATA_* definition.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn or user is NULL, or mod_user has invalid role.
OPC_ERR_INVALID_OUTPARAM	mod_user is NULL or invalid.

OPC_ERR_ACCESS_DENIED	User has no administrator rights.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_USER_NOT_FOUND	User not found.
OPC_ERR_LOCKED_BY_OTHER	This object is currently locked.
OPC_ERR_DEADLOCK	Deadlock situation detected.
OPC_ERR_OBJECT_ALREADY_EXISTS	Name of mod_user already exists.
OPC_WARN_PAM_ENABLED	User modified successfully but password may be ineffective because PAM is enabled.

Versions

5.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 586

“opc_connect()” on page 181

Distribution API

The Distribution API provides a function to distribute HP Operations agent configuration to managed nodes.

Data Structures

OPCDTYPE_CONTAINER

OPCDTYPE_NODE_CONFIG

Usage

The Distribution API can only be called with administrator rights. In addition, it can be called only with root permissions because of access restrictions of the distribution directories.

Prerequisites

The Distribution API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the Distribution API are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

opc_distrib()

```
#include opcsvapi.h

int opc_distrib (
              opc_connection      opc_conn,      /* in */
              int32               components,    /* in */
              bool                force_update,  /* in */
              opcdata             nodeconf_list  /* in */
              );
```

Parameters

`opc_conn` Connection to the HPOM database.

`components` Is a bitmask field; the following flags are available:

- `OPC_DISTRIB_TEMPLATES`
- `OPC_DISTRIB_ACTIONS`
- `OPC_DISTRIB_MONITORS`
- `OPC_DISTRIB_COMMANDS`
- `OPC_DISTRIB_ALL`

`force_update` If set to `FALSE`, only the components that have been changed are distributed; if set to `TRUE`, all specified components are distributed.

`nodeconf_list` Container of type `OPCDTYPE_NODE_CONFIG`.

Description

Use the function `opc_distrib()` to distribute HP Operations agent configuration to managed nodes. This API offers the same functionality as when a user distributes policies, actions, monitors, and commands from the `Install / Update HPOM Software and Configuration` window. It cannot be used to install the HP Operations agent software on managed nodes.

Only monitored and/or controlled nodes can be used as target for the distribution. Other types of node types, for example external nodes are ignored.

Return Values

OPC_ERR_OK	No error occurred.
OPC_ERR_INVALID_INPARAM	One of the input parameters is NULL or not of the correct type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation error.
OPC_ERR_ACCESS_DENIED	Application is not called as HPOM administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while `opccfgupld` is running.
OPC_ERR_CANT_GET_MGMTSV_ADDRESS	The address of the management server cannot be retrieved.
OPC_ERR_NOT_COMPLETELY_DONE	Not all objects in `nodeconf_list` were processed.
OPC_WARN_EMPTY_CONTAINER	No nodes were specified.

Versions

5.00 and later

Server Synchronization API

The Server Synchronization API provides functions that allow you to control access and modification of the HPOM configuration data.

Since multiple processes are able to manipulate HPOM configuration data, there is a need to control both the configuration data and its modification. In HPOM, applications lock data to avoid the risk of concurrent modifications by other applications or users. However, after modification, the server processes and the user interfaces need to be synchronized to see and use the new configuration data. Locking the data aims to avoid the scenario where several processes manipulate each others' changes. See "The Transaction Concept" on page 501 for more detailed information about the way in which HPOM synchronizes changes to configuration data.

Data Structures

OPCDTYPE_INFORM_USER

OPCDTYPE_USER_CONFIG

Usage

The Server Synchronization API can be called by any user. To use the functions, include the header file `opcsvapi.h` in your application.

Prerequisites

The Server Synchronization API is only available on the management server.

Multithread Usage

All functions of the HPOM Server Synchronization APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads and Kernel Threads. They are neither async-cancel, async-signal, nor fork-safe.

The Transaction Concept

HPOM's transaction concept requires API functions to start, commit, and rollback user transactions and uses Oracle transactions and Oracle locks, which are already available. However, only single objects are locked, not complete object types nor any dependents. In addition, objects are only locked if they are to be modified (and ideally immediately before). This reduces restrictions and avoids functions locking each other out.

After a user transaction (commit or rollback) starts, all locks are cumulated until the user transaction is finished. If no user transaction is set, the database functions use a database transaction and set locks for modifying database access. The database transaction and corresponding locks are released before the API function exits. This ensures that locks relating to other user transactions are respected. The timeout concept used elsewhere in HPOM is also used here, namely; if HPOM cannot get a lock, it waits five (5) seconds before trying again. It tries to get the lock three (3) times. It is recommended that user transactions be as short as possible. Long transactions increase the possibility of lock conflicts and deadlocks with other processes and require a lot of resources (rollback segments of the database).

NOTE Existing Oracle tools may be used to troubleshoot locking problems.

HPOM ensures referential integrity in the database. The user-transaction concept gives the API user the ability to ensure logical integrity. If several API function calls modify the same object, and one of the API calls subsequently fails, all modifications can be rolled back.

Transaction Rules

The behavior of HPOM's configuration API functions is determined by rules which are applied according to how the functions are called:

❑ **General rules**

- API functions that get a list of objects do *not* take a lock.
- All modifying API functions (_add, _modify, _delete) lock the object to be modified.
- If an API function is unable to get a lock on an object, it returns the error code `OPC_ERR_LOCKED_BY_OTHER`.
- The same timeout concept used elsewhere in HPOM is also used here. If a function is unable to get a lock, it waits five (5) seconds and tries again. It retries three (3) times.

❑ **API functions called within a user transaction**

- A function which reads a single object locks the object.
- Locks are cumulated until the user transaction is complete.
- If an error occurs during a transaction, the changes made during the failed transaction are rolled back to ensure database consistency. However, it is not necessarily true that *all* changes in this user transaction are *automatically* rolled back. In addition, the API user can decide to continue in spite of this error or roll back the complete transaction.
- If an API function encounters a deadlock, it returns the error code `OPC_ERR_DEADLOCK` and rolls back only those changes made by the current API function. However, it is recommended the complete user transaction be rolled back since the reason for the deadlock may lie in an earlier lock.

❑ **API functions called when no user transaction is open**

- A function which reads a single object does *not* lock it. It is possible to read objects that are locked by another process.
- Each API function has its own database transaction. Locks are taken, but they are released before the API function returns.
- If a deadlock is encountered, the API function returns the error code `OPC_ERR_DEADLOCK`. The changes are rolled back.

opc_inform_user()

```
#include opcsvapi.h

int opc_inform_user (
            const  opc_connection    opc_conn,    /* in */
            opcdata                  user_info    /* in */
                   );
```

Parameters

`opc_conn` Connection to the HPOM database.

`user_info` Data object of type `OPCDTYPE_INFORM_USER` that contains the following fields, which may be modified with `opcdata_set_str()`:

- `OPCDATA_TEXT`

  Text of the message sent to the user interfaces.

- `OPCDATA_NAME`

  Name of an HPOM user; if undefined, all HPOM users are informed.

Description

`opc_inform_user()` is a generic function which allows users and applications to supply user interfaces with information. This function is not dependent on previous changes and may be used to inform users before or after making any modifications, or simply to send a message to an HPOM user. This API function can be also called by API users without administrator permissions.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL.
`OPC_ERR_CANT_CONNECT_DM`	Access to display manager failed.
`OPC_ERR_NO_MEMORY`	Memory allocation failed.

Versions

5.00 and later

opc_version()

If used on the *management server*:

```
#include opcsvapi.h

char *  opc_version ();
```

If used on the *managed node*:

```
#include opcapi.h

char *  opc_version ();
```

Description

Returns the `what` string of the HPOM library that is used in this version.

Return Values

The returned string has the format:

`HP <product> <version> (<date>)`

For example:

HP Operations Manager 8.10 (03/16/07).

Versions

6.00 and later

See Also

what(1)

opcsync_inform_server()

```
#include opcsvapi.h

int opcsync_inform_server (
          const opc_connection opc_conn         /* in */
                );
```

Parameters

opc_conn	Connection to the HPOM database.

Description

Synchronizes the server and GUI processes with any configuration changes performed since startup or the last execution of this function. Although synchronization time is reduced overall, server-process data may be out-of-date (and messages forwarded to wrong users) for a short time. If the application exits without calling opcsync_inform_server(), the server processes are not informed until the next restart.

NOTE opcsync_inform_server() is automatically called at the end of an HPOM session (within opc_disconnect()). Therefore, it may not be necessary to call it actively.

NOTE opcsync_inform_server() informs the server and GUI processes of new configuration changes. The HPOM server and GUIs must be kept up-to-date each time a change has been performed. There is no need to reload the configuration with each change in most cases. However, some configuration changes still require Java GUI reload, see the *HPOM Java GUI Operator's Guide* for details.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_CANT_CONNECT_DM	Access to display manager failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.

OPC_ERR_ACCESS_DENIED	Not connected as administrator.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“opc_disconnect()” on page 183

opctransaction_commit ()

```
#include opcsvapi.h

int opctransaction_commit (
          const opc_connection       opc_conn   /* in */
                        )
```

Parameters

opc_conn	Connection to the HPOM database.

Description

Finishes the current user transaction and commits all changes associated with this transaction. The cumulated locks are released.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_ACCESS_DENIED	Not connected as administrator.
OPC_ERR_NO_TRANSACTION	No transaction is open.

Versions

5.00 and later

opctransaction_rollback()

```
#include opcsvapi.h

int opctransaction_rollback (
          const opc_connection        opc_conn   /* in */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.

Description

Finishes the current user transaction and rolls back (undoes) all changes to this transaction. Any cumulated locks are released.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_ACCESS_DENIED	Not connected as administrator.
OPC_ERR_NO_TRANSACTION	No transaction is open.

Versions

5.00 and later

opctransaction_start ()

```
#include opcsvapi.h

int opctransaction_start (
          const opc_connection        opc_conn    /* in */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.

Description

Starts a user transaction; locks are cumulated during the user transaction.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_ACCESS_DENIED	Not connected as administrator.
OPC_ERR_TRANSACTION_ALREADY_OPEN	Transaction is already open, transaction has started.

Versions

5.00 and later

Pattern Matching API

The Pattern Matching API provides a function to match a string against a pattern.

Usage

The Pattern Matching API can only be called by any user.

Prerequisites

The Pattern Matching API is only available on the management server. To use the functions, include the header file `opcsvapi.h` in your application.

Multithread Usage

All functions of the HPOM Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads. They cannot be safely called in kernel threads and are neither async-cancel safe, async-signal safe, nor fork-safe.

opcpat_match()

```
#include opcsvapi.h

int opcpat_match (
   const  char *  str,         /* in */
   const  char *  pattern,     /* in */
   const  char *  separator,   /* in */
   int            cflag,       /* in */
   int    *       pmres,       /* out */
   char   ***     vars,        /* out */
   char   ***     vals,        /* out */
   int    *       len          /* out */
   );
```

Parameters

`str`	Input string.
`pattern`	HPOM pattern expression.
`separator`	Field separator. If `separator` is set to NULL, the default separator is used.
`cflag`	Case-sensitive mode: 0—Case-sensitive mode 1—Case-insensitive mode

`pmres`	Result of match operation:
	`OPC_PATTERN_MATCHES`
	Successful match operation
	`OPC_PATTERN_MATCHES_NOT`
	Unsuccessful match operation

`vars`	Returned pointers to the list of assigned variables.
`vals`	Returned pointers to the list of values of the assigned variables.
`len`	Number of elements in each list.

Description

`opcpat_match()` returns the result of an HPOM pattern-matching operation and lists the matched strings, assigned to variables. The caller must provide an input string, a pattern, a separator, and the case-sensitive mode.

`opcpat_match()` also returns the results of assign-to-variable operations. The lists `vars` and `vals` hold variables and theirs values, respectively.

NOTE The caller must free the memory of all elements in both lists and the lists themselves.

Return Values

`OPC_ERR_OK`	No error occurred.
`OPC_ERR_INVALID_INPARAM`	`vars` is NULL, `vals` is NULL, `pattern` is invalid.
`OPC_ERR_NO_MEMORY`	Memory allocation error.

Versions

8.26 and later

4 Examples

In this Chapter

This chapter provides examples explaining how you can use the HPOM APIs.

- ❏ Examples of the HPOM Interfaces
  - Using the Server MSI API to Connect to the HPOM Interface
  - Registering for Message Events from the HPOM Interface
  - Using the Configuration Stream Interface (CSI)
  - Example of the Legacy Link Interface API, Part 1
  - Example of the Legacy Link Interface API, Part 2
- ❏ Example of the Server Message API

Examples

Example programs and the corresponding makefiles for each supported managed node platform are provided in the directory `/opt/OV/OpC/examples/progs` on the management server. See also the `README` file in the same directory.

Examples of the HPOM Interfaces

Using the Server MSI API to Connect to the HPOM Interface

The following program connects to the Server Message Stream Interface as a read-write application. It scans for `DISK_FULL` messages from a node `omnisv`, and then generates a message `OMNIBACK_CANT_WRITE_LOG` message after a period of five minutes.

```
void main()
{
    opcdata msg;
    opcregcond reg_cond;
    int if_id;
    long t_disk_full = 0;

    if (opcif_open(OPCSVIF_EXTMSGPROC_READWRITE,
     "MyInterface", OPCIF_SV_RUNNING | OPCIF_READ_WAIT, 0,
     &if_id))
    {
        exit(1);
    }

    /* set up register conditions */
    if (opcreg_create(&reg_cond))
    {
        exit(1);
    }
    /* type DISK_FULL && node "omnisv"
    if (opcreg_set_str(reg_cond, OPCREG_MSGTYPE,
     "DISK_FULL") ||
     opcreg_set_str(reg_cond, OPCREG_NODENAME, "omnisv"))
    {
        exit(1);
```

```
    }
    if (opcif_register(if_id, reg_cond, NULL))
    {
        exit(1);
    }
    /* type OMNIBACK_CANT_WRITE_LOG && node "omnisv"
    if (opcreg_set_str(reg_cond, OPCREG_MSGTYPE,
     "OMNIBACK_CANT_WRITE_LOG"))
    {
        exit(1);
    }
    if (opcif_register(if_id, reg_cond, NULL))
    {
        exit(1);
    }

    /* create empty message */
    if (opcdata_create(OPCDTYPE_EMPTY, &msg))
    {
        exit(1);
    }

    /* read and process incoming messages */
    for ( ; ; )
    {
        if (opcif_read(if_id, msg))
        { /* signal received or read error */
            break;
        }

        if (!strcmp(opcdata_get_str(msg,
         OPCREG_MSGTYPE), "DISK_FULL"))
        {
            t_disk_full = opcdata_get_long(msg,
            OPCDATA_CREATION_TIME)
        }
        else if (!strcmp(opcdata_get_str(msg,
OPCREG_MSGTYPE),
         "OMNIBACK_CANT_WRITE_LOG"))
        {
            if (t_disk_full != 0 && opcdata_get_long(msg,
             OPCDATA_CREATION_TIME) - t_disk_full < 5*60)
```

```
            {
                /* no error handling here: */
                opcdata_set_str(msg, OPCDATA_MSGTEXT,
                 "Omniback cannot write logfile because \
                 disk is full");
            }
        }

        opcif_write(if_id, msg);
    } /* for */

    opcif_close(if_id);
    exit(0);

} /* main() */
```

Registering for Message Events from the HPOM Interface

This program registers for all Message Events (MEs) from the OPCSVIF_MSG_EVENTS queue, including information message events sent with opcwall. For every incoming message event, it will print out the message ID, the event flag and the time received.

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

#include "opcsvapi.h"


int main (int argc, char *argv[])

{
  int             ret, i, read_again;
  int             if_id;
  long            cond_id1;
  long            cond_id2 = 0;
  char            string[32];
  char            opername[32];
  char            iname[32];
  int             ttime;
  opcregcond      reg_cond1;
  opcregcond      reg_cond2;
  opcdata         event;

  /* Enter some important things */
  printf ("Enter Instance Name: ");
  gets (iname);
  printf ("Enter testing time in seconds: ");
  gets (string);
  ttime = atoi (string);
  printf ("Enter operator name or 'Enter' for none: ");
  *opername = '\0';
  gets (opername);
  printf ("\n\n");

  if ( (ret = opcreg_create (&reg_cond1)) != OPC_ERR_OK )
  {
```

```
    fprintf (stderr, "Unable to create registration
    condition!\n");
    fprintf (stderr, "opcreg_create() returned with %d\n",
ret);
    fflush (stderr);
    goto error_exit;
  }

  if ( (ret = opcreg_create (&reg_cond2)) != OPC_ERR_OK )
  {
    fprintf (stderr, "Unable to create registration
    condition!\n");
    fprintf (stderr, "opcreg_create() returned with %d\n",
ret);
    fflush (stderr);
    goto error_exit;
  }

  /* open interface for all MEs */
  printf ("Open instance '%s'\n", iname);

  if ( (ret = opcif_open (OPCSVIF_MSG_EVENTS,
                          iname,
                          OPCIF_ALWAYS | OPCIF_READ_NOWAIT,
                          100,
                          &if_id))

       != OPC_ERR_OK )
  {
    fprintf (stderr, "Unable to open instance for MEs!\n");
    fprintf (stderr, "opcif_open() returned with %d\n",
ret);
    fflush (stderr);
    goto error_exit;
  }

  /* register for all MEs */
  printf ("Register for all MEs\n");

  if ( (ret = opcreg_set_long (reg_cond1,
                               OPCREG_MSG_EVENT_MASK,
                               (long) OPC_MSG_EVENT_ALL))
```

```
      != OPC_ERR_OK )
  {
    fprintf (stderr, "Unable to set registration
condition!\n");
    fprintf (stderr, "opcreg_set_long() returned with %d\n",
ret);
    fflush (stderr);
    goto error_exit;
  }

  if ( (ret = opcif_register (if_id, reg_cond1, &cond_id1))
  != OPC_ERR_OK )
  {
    fprintf (stderr, "Unable to register registration
    condition!\n");
    fprintf (stderr, "opcif_register() returned with %d\n",
ret);
    fflush (stderr);
    goto error_exit;
  }

  /* If an operator name is specified, set additional
condition */
  if ( strlen (opername) != 0 )
  {
    printf ("Register for MEs only for operator '%s'\n",
    opername);

    if ( (ret = opcreg_set_str (reg_cond2, OPCREG_OPERATOR,
       opername))!= OPC_ERR_OK )
    {
      fprintf (stderr, "Unable to set registration condition
         for operator!\n");
      fprintf (stderr, "opcreg_set_long() returned with
%d\n",
        ret);
      fflush (stderr);
      goto error_exit;
    }

    if ( (ret = opcif_register (if_id, reg_cond2,
&cond_id2))
```

```
      != OPC_ERR_OK )
    {
      fprintf (stderr, "Unable to register registration
        condition for operator!\n");
      fprintf (stderr, "opcif_register() returned with
%d\n",
      ret);
      fflush (stderr);
      goto error_exit;
    }
  }

  if ( (ret = opcdata_create (OPCDTYPE_MESSAGE_EVENT,
&event))
    != OPC_ERR_OK )
  {
    fprintf (stderr, "Cannot create empty event!\n");
    fprintf (stderr, "opcdata_create() returned with %d\n",
ret);
    fflush (stderr);
    goto error_exit;
  }

  printf ("Waiting for MEs for the next %d seconds \n",
ttime);
  read_again = 1;

  while (read_again)
  {
    /* receive the events and print it to stdout */
    for (i = 0; i < ttime; i++)
    {
      sleep (1);

      if ( (ret = opcif_read (if_id, event)) != OPC_ERR_OK
&&
         ret != OPC_ERR_NO_DATA )
      {
        fprintf (stderr, "Unable to read Message Change
          Event!\n");
        fprintf (stderr, "opcif_read() returned with %d\n",
ret);
```

```
        fflush (stderr);
        goto error_exit;
      }
      else if ( ret == OPC_ERR_OK )
      {
        fprintf (stdout, "Message Event Flag:\t0x%lx\n",
                 opcdata_get_long (event,
OPCDATA_EVENT_FLAG));
        fprintf (stdout, "Message ID:\t\t%s\n",
                 opcdata_get_str (event, OPCDATA_MSGID));
      }
    }

    printf ("Do you want to read MEs for another %d seconds?
",
      ttime);
    gets (string);
    if (*string == 'n') read_again = 0;
    printf ("\nOK, waiting for more MEs!\n");
  } /* while */

  ret = OPC_ERR_OK;

error_exit:

  opcdata_free (&event);

  opcreg_free (&reg_cond1);
  opcreg_free (&reg_cond2);

  printf ("Unregister for MEs\n");
  opcif_unregister (if_id, cond_id1);
  if ( cond_id2 )
    opcif_unregister (if_id, cond_id2);

  printf ("Close Instance %s\n", iname);
  opcif_close (if_id);
opcreg_free (&reg_cond);
  return (ret);
}
```

Using the Configuration Stream Interface (CSI)

To start using CSI, you first need to connect to the database and then open the interface. You can set a OPCDATA_CSI_STRING to identify yourself. This is used to avoid getting back your own configuration changes. Provide the same name of the CSI interface as used in opcif_open() call.

```
int             itfc_id;
opc_connection  conn;
…
opc_connect("opc_op",<passwd>,&conn);
opcconn_set_capability(connection,
OPCDATA_CSI_STRING,"my_itfc");
…
opcif_open (OPCSVIF_CFG_CHG_EVENTS, "my_itfc", OPCIF_ALWAYS
| OPCIF_READ_WAIT | OPCIF_CLOSE_FORWARD, 100, &itfc_id);
…
```

To register for one or several configuration change events, use OPCREG_MSG_EVENT_MASK. To register for all configuration change events, use OPC_CCE_EVENT_ALL. All configuration change events are listed in the Table 5-9 on page 555.

For OPCSVIF_CFG_CHG_EVENTS_GUI, set OPCREG_OPERATOR to receive only the configuration change events that this operator is allowed to see.

```
int             itfc_id;
opcregcond      reg_cond = NULL;
int             cond_id;
/* first open the interface */
opcif_open (OPCSVIF_CFG_CHG_EVENTS_GUI, "my_itfc",
OPCIF_ALWAYS | OPCIF_READ_WAIT | OPCIF_CLOSE_FORWARD, 100,
&itfc_id);
/* create reg. conds: register for opc_op and all events */
opcreg_create (&reg_cond);
opcreg_set_str (reg_cond, OPCREG_OPERATOR, "opc_op");
opcreg_set_long (reg_cond, OPCREG_MSG_EVENT_MASK,
OPC_CCE_EVENT_ALL);
opcif_register (itfc_id, reg_cond, &cond_id)
/* cleanup */
opcreg_free (&reg_cond);
;
```

You can read configuration change events arriving to CSI in `opcdata` format using the `opcif_read()`.

```
opcdata          cce      = NULL;
int              type;
char * obj_id    = NULL;
/* read cce from interface */
opcdata_create (OPCDTYPE_CONFIG_EVENT, &cce);
opcif_read (itfc_id, cce));
/* get attributes from CCE */
type=opcdata_get_long(cce,OPCDATA_OBJECT_TYPE);
obj_id=opcdata_get_str(cce,OPCDATA_ID);
printf("Received object of type %d with id %s.\n", type,
obj_id);
```

Example of the Legacy Link Interface API, Part 1

This example submits messages from a legacy system to the internal message stream of the management server. The function calls are submitted by the process using the API.

```
#include <stdio.h>
#include <signal.h>
#include <opcsvapi.h>

#define FALSE (0)
#define TRUE (1)

extern void display_msg (opcdata data);
void close_if_and_exit(int exitstatus);
void sighandler();

int interface_id = 0;
int main (int argc, char * argv[])
{

int ret;

     int interface_type
     char instance_name[] = "myLegacyIf";
     int mode;

     char msgText[256];
     opcdata msg;

     interface_type = OPCSVIF_EXTAGT_MESSAGE;
     mode = OPCIF_ALWAYS;
     ret = opcif_open(interface_type, instance_name, mode,
0,
     &interface_id);
     if (ret != OPC_ERR_OK) { ... }

     /* define a signal handler to close the interface in
case the
     program is terminated by a SIGTERM or a SIGNINT signal
*/

     (void) signal (SIGTERM, sighandler);
```

```
     (void) signal (SIGINT, sighandler);

     for (;;)
     {
          /* read message text from stdin */

          printf("Please enter message text: ");
          (void) fgets(msgText, 256, stdin);

          /* format a message using the message text */
          ret = opcdata_create(OPCDTYPE_MESSAGE, &msg);
          if (ret != OPC_ERR_OK) { ... }

          opcdata_set_str(msg, OPCDATA_MSGTEXT, msgText);

          /* write message to the management server and
display it
          on standard output */

          ret = opcif_write(interface_id, msg);
          if (ret != OPC_ERR_OK) { ... }

          printf("The following message was sent:\n");
          display_msg(msg);

          (void) opcdata_free(&msg);
     }
} /* end main */

void close_if_and_exit(int exitstatus)
{
     int ret;

     ret = opcif_close(interface_id);
     if (ret != OPC_ERR_OK) { ... }}

void sighandler()
{
     close_if_and_exit(0);
}
```

Example of the Legacy Link Interface API, Part 2

This example extends Part 1 above, to receive action requests and to send action responses. The function calls are submitted by the process using the API.

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <opcsvapi.h>


extern void display_msg    (opcdata data);
extern void display_actreq (opcdata data);
extern void display_actresp(opcdata data);

void close_if_and_exit(int exitstatus);
void sighandler();
void handle_legacy_input();
void handle_action_request();

int msgIF_id = 0;
int actReqIF_id = 0;
int actRespIF_id = 0;

int main (int argc, char * argv[])
{
     int ret;
     int rc;

     int interface_type;
     char* instance_name;
     int mode;

     struct fd_set readfds;
     int pipefd;

     long cond_id;

     interface_type = OPCSVIF_EXTAGT_MESSAGE;
     mode = OPCIF_ALWAYS;
```

```
     instance_name = "legMsgIF";

     ret = opcif_open(interface_type, instance_name, mode,
0,
               &msgIF_id);
     if (ret != OPC_ERR_OK) { ... }

     interface_type = OPCSVIF_EXTAGT_ACTION_REQUEST;
     mode = (OPCIF_ALWAYS | OPCIF_READ_WAIT);
     instance_name = "legActReqIF";

     ret = opcif_open(interface_type, instance_name, mode,
0,
               &actReqIF_id);
     if (ret != OPC_ERR_OK) { ... }

     interface_type = OPCSVIF_EXTAGT_ACTION_RESPONSE;
     mode = OPCIF_ALWAYS;
     instance_name = "legActRespIF";

     ret = opcif_open(interface_type, instance_name, mode,
0,
          &actRespIF_id);
     if (ret != OPC_ERR_OK) { ... }

     /* define a signal handler to close the interfaces in
     case the program is terminated by a  SIGTERM or a
     SIGNINT signal */

     (void) signal (SIGTERM, sighandler);
     (void) signal (SIGINT, sighandler);
     ret = opcif_get_pipe(actReqIF_id, &pipefd);

     if (ret != OPC_ERR_OK) { ... }

     for (;;)
     {
          printf("If you want a message to be sent, ");
          printf("please enter message text.\nText: ");

          /* issue a select() call for listening
simultanously at
```

```
          standard input and at the action request interface
*/

          FD_ZERO(&readfds);
          FD_SET (STDIN_FILENO, &readfds);
          FD_SET (pipefd, &readfds);

          rc = select(NFDBITS, (int*) &readfds, (int*) 0,
(int*)
               0,
          (struct timeval *) 0);

          if (rc == -1) { ... }

          /* check whether an action request from the
management
          server arrived */

          if (FD_ISSET(pipefd, &readfds))
          {
               handle_action_request();
          }
          if (FD_ISSET(STDIN_FILENO, &readfds))
          {
               handle_legacy_input();
          }
     } /* end for */
} /* end main */
void handle_legacy_input()
{
     int ret;
     char msgText[256];
     opcdata msg;
     /* read message text from standard input and generate a
     message */

     (void) fgets(msgText, 256, stdin);
     ret = opcdata_create(OPCDTYPE_MESSAGE, &msg);
     if (ret != OPC_ERR_OK) { ... }

     opcdata_set_str(msg, OPCDATA_MSGTEXT, msgText);
     ret = opcif_write(msgIF_id, msg);
```

```
        if (ret != OPC_ERR_OK) { ... }

        printf("The following message was sent:\n");
        display_msg(msg);

        (void) opcdata_free(&msg);
}

void handle_action_request()
{
        int ret;
        opcdata actReq;
        opcdata actResp;

        ret = opcdata_create(OPCDTYPE_EMPTY, &actReq);
        if (ret != OPC_ERR_OK) { ...}

        ret = opcif_read(actReqIF_id, actReq);
        if (ret != OPC_ERR_OK) { ... }

        printf("The following action request was received:\n");
        display_actreq(actReq);

        /* prepare a response to the action request */

        ret = opcdata_create(OPCDTYPE_ACTION_RESPONSE,
&actResp);
        if (ret != OPC_ERR_OK) { ... }

        ret = opcdata_copy_info_to_actresp(actReq, actResp);
        if (ret != OPC_ERR_OK) { ... }

        opcdata_set_str(actResp, OPCDATA_ACTION_OUTPUT,
        "This is the output of the action requested by you");

        ret = opcif_write(actRespIF_id, actResp);
        if (ret != OPC_ERR_OK) { ... }

        printf("The following action response was sent:\n");
        display_actresp(actResp);

        (void) opcdata_free(&actReq);
```

```
     (void) opcdata_free(&actResp);
}
void close_if_and_exit(int exitstatus)
{
     int ret;

     /* close all three the interfaces */

     ..........

     exit(exitstatus);
}
void sighandler()
{
     close_if_and_exit(0);
}
```

Example of the Server Message API

```
/* example of the Server Message API - with error checking */
/* Usage:  <progname> <message-id>*
   Given a list of message IDs (as strings) this application
prints
   the message details to stdout
*/
#include <opcsvapi.h>

char *operator="opc_adm";
char *password="OpC_adm";

/*-------------------------------------------------------------
----
* Function: report_error
* ----------------------
* Description:
*       Maps an HPOM error code to an error text and prints
this *       text to stderr.
*
* Input Parameters:
*       err_code           HPOM error code.
*
* Output Parameters:
*       none.
*
* Errors:
*       none.
*
* Return values:
*       none.
*
*-------------------------------------------------------------
-----
*/

static void report_error(err_code)
          int err_code;
{
          static struct err_tbl_entry {
               int  e_code;
```

```
                char *err_text;
            } err_tbl[] = {
            { OPC_ERR_OK, "No error." },
            { OPC_ERR_NO_MEMORY, "Insufficient memory." },
            { OPC_ERR_CANT_INIT, "HPOM error: Can't
initialize." },
            { OPC_ERR_ACCESS_DENIED, "You are not allowed to
perform
                this operation" },
            { OPC_ERR_INVALID_OPERATOR, "The operator is not
defined
                in the HPOM database" },
            { OPC_ERR_INVALID_PASSWORD, "Invalid password" },
            { OPC_ERR_NO_LOGIN, "Login failed" },
            { OPC_ERR_CANT_CONNECT_DB, "Cannot connect to DB"
},
            { OPC_ERR_NO_OPERATOR_DEF, "No definition for
operator
                found"},
            { OPC_ERR_NOT_CONNECTED, "Not connected to HPOM
                database" },
            { OPC_ERR_CANT_DISCONNECT, "Cannot disconnect from
                HPOM" },
            { OPC_ERR_INVALID_OUTPARAM, "Output parameter is
NULL"},
            { OPC_ERR_INVALID_STRUCTURE, "Given structure is
not
                initialized"},
            { OPC_ERR_NOT_FREED, "Unable to free allocated
memory"},
            { OPC_ERR_INVALID_ID, "Given message ID is
malformed"},
            { OPC_ERR_NO_MESSAGE, "Message with given ID does
not
                exist"},
            { OPC_ERR_FUNCTION_FAILED, "Function failed"},
            { OPC_ERR_NO_ANNOTATIONS, "The message has no
                annotations"},
            { OPC_ERR_ALREADY_DONE, "The action was already
                performed or the message was already
acknowledged"}
            };
```

```
          int i;
          /* Look for the error code and print the
corresponding              error text. */

          for(i = 0; i < (sizeof(err_tbl) / sizeof(struct
               err_tbl_entry));
i++)
               {
                    if(err_code == err_tbl[i].e_code)
                         {
                              fprintf(stderr,
                                   "%s\n",
err_tbl[i].err_text);
                              return;
                         }
               }
          /* Unknown error code. */

          fprintf(stderr, "Unknown error: %d\n", err_code);
          return;
}

main(argc, argv)
          int argc;
          char **argv;
{
          int                    ret, i;
          opc_connection         opc_conn;
          opcdata                data;
          opc_annotations        annotations;
          opcdata                message_id;

          /* first connect to the HPOM DB as opc_adm */
          if ((ret = opc_connect(operator, password,
&opc_conn))
                    != OPC_ERR_OK)
               {
                    report_error(ret);
                    exit(1);
               }  /* if */
          /* create different structures */
```

```
                if((ret=opcdata_create(OPCDTYPE_MESSAGE_ID,
                    &message_id))
                          != OPC_ERR_OK)
                     {
                          report_error(ret);
                          exit(1);
                     }
                if((ret=opcdata_create(OPCDTYPE_MESSAGE,&data))
                          != OPC_ERR_OK)
                     {
                          report_error(ret);
                          exit(1);
                     }
                /* loop over all arguments */
                for(i=1; i<argc; i++)
                     {
                          /* convert the string notation of the ID
into
                               internal format */
                          if((ret=opcdata_set_string(argv[i],
                              message_id))
                               != OPC_ERR_OK)
                     {
                          report_error(ret);
                          continue;
                     }

                /* get the message details */
                if((ret=opcmsg_get(opc_conn, message_id, data))
                    !=OPC_ERR_OK)
                     {
                          report_error(ret);
                          continue;
                     }
                /* present them */
                printf("Message text: %s\n",
                     opcdata_get_str(data, OPCDATA_MSGTEXT));

                /* now get the instructions */
                printf("Instructions: %s\n",
                     opcmsg_get_instruction(opc_conn,
message_id));
```

```
                /* get all annotations */
                if((ret=opcdata_create(OPCDTYPE_ANNOTATION,
                     &annotations)) != OPC_ERR_OK)
                     {
                          report_error(ret);
                          continue;
                     }
                if((ret=opcmsg_get_annotations(opc_conn,
message_id,
                     annotations)) != OPC_ERR_OK)
                     {
                          report_error(ret);
                          continue;
                     }
                /* print all annotations */
                for(i=1;i<opcanno_get_count(annotations);i++)
                     {
                          printf("%d. Annotation:\
                          %s\n",opcanno_get(annotations,i));
                     }
                /* free annotations */
                if((ret=opcanno_free(&annotations))!=OPC_ERR_OK)
                     {
                          report_error(ret);
                     }

                /* if operator action defined, then start it: */
                if(strcmp(opcdata_get_str(data,
                     OPCDATA_OPACTION_CALL),"") == 0)
                     if((ret=opcmsg_start_op_action(opc_conn,
                          message_id)) != OPC_ERR_OK)
                          report_error(ret);

                /* add an annotation */
                if((ret=opcmsg_add_annotation(opc_conn,
message_id,
                     "blablabla")) != OPC_ERR_OK)
                     report_error(ret);

                /* acknowledge message */
                if((ret=opcmsg_ack(opc_conn, message_id))!=
```

```
OPC_ERR_OK)
                report_error(ret);
     }

          /* free resources */
          if((ret=opcid_free(&message_id)) !=OPC_ERR_OK)
               report_error(ret);
          if((ret=opcmsg_free(&data)) != OPC_ERR_OK)
               report_error(ret);

          /* disconnect from HPOM DB */
          if((ret=opc_disconnect(&opc_conn)) != OPC_ERR_OK)
               report_error(ret);
}
```

Examples

5 HPOM Data Structures

In This Chapter

This chapter lists the HPOM data structures and their attributes.

The tables in this chapter use the following structure:

Attribute The first column contains the attributes of a data structure.

Scope The second column defines the scope of an attribute: `get` or `set`.

Type The third column defines the type of an attribute: `long` or `string[len]`. For each string the maximal length of the string is given.

Properties The fourth column contains the following additional parameters:

`R`	The attribute is required for set or get functions.
`K`	Key for get functions.
`L`	The attribute can be localized.

Description The fifth column contains a description of the attribute.

HPOM Data Structures

HPOM provides a set of data structures which hold information about HPOM objects.

The attributes of each of these data types are described in the following tables. The tables show which of the attributes can only be retrieved and which can be set.

- Functions to retrieve attributes
  - opcdata_get_long()
  - opcdata_get_str()
  - opcdata_get_double()
  - opcdata_lget_len()
  - opcdata_lget_long()
  - opcdata_lget_str()
- Functions to set attributes
  - opcdata_set_long()
  - opcdata_set_str()
  - opcdata_set_double()
  - opcdata_lset_long()
  - opcdata_lset_str()

The description also includes information about the type of the attribute value (long, string, or double), and, if the attribute value is a string, the maximum character length (for example, `str[32]`).

The available predefined values are defined in the include files `/opt/OV/include/opcapi.h` and `/opt/OV/include/opcsvapi.h`.

You can also see the man page *opcdata(3)* for more information.

OPCDTYPE_CONTAINER

Container elements are only accessible by way of the `opcdata_*_element()` functions.

OPCDTYPE_ACTION_REQUEST

Table 5-1 lists the attributes that are available for the Action Request data structure.

Table 5-1 OPCDTYPE_ACTION_REQUEST

Attribute	Scope	Type	Properties	Description
OPCDATA_ACTION_TYPE	get	long		Defines the type of HPOM action: • OPC_AUTOMATIC_ACTION • OPC_OPERATOR_INIT_ACTION • OPC_TERMINAL_APPLICATION • OPC_NO_TERMINAL_APPLICATION • OPC_GET_INSTRUCTION_TEXT
OPCDATA_ACTION_ ANNOTATE	get	long		For actions of type OPC_AUTOMATIC_ACTION and OPC_OPERATOR_INIT_ACTION this attribute defines whether the HP Operations management server is required to create an annotation for the message that triggered the action. Possible values are: 0 (default): don't create an annotation 1: create an annotation after execution
OPCDATA_ACTION_ACK	get	long		For actions of type OPC_AUTOMATIC_ACTION and OPC_OPERATOR_INIT_ACTION this attribute defines whether the HP Operations management server acknowledges the message that triggered the action after the action was executed successfully. Possible values are: 0 (default): do not auto-acknowledge 1: auto-acknowledge
OPCDATA_NODENAME	get	str		Name of the node on which the action should be started.
OPCDATA_ACTION_CALL	get	str	L	Action command to execute.
OPCDATA_ACTION_USER	get	str		User under whose permissions the action will be started.

Table 5-1 OPCDTYPE_ACTION_REQUEST (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_ACTION_PWD	get	str		Password f the action user on the system. If this attribute is provided as empty string, no password check is necessary. The action must be executed using the defined user. If a password is provided for the attribute the agent must first check whether the password is correct for the defined user.
OPCDATA_DISPLAY	get	str		Defines the display of the HPOM GUI that sends the action request. This display value is important to start X-applications. The agent must use this display before the application is started to define that windows of the application are displayed on the correct screen.

OPCDTYPE_ACTION_RESPONSE

Table 5-2 lists the attributes that are available for the Action Response data structure.

Table 5-2 OPCDTYPE_ACTION_RESPONSE

Attribute	Scope	Type	Properties	Description
OPCDATA_ACTION_TYPE	get/set	long		Type of the action: • OPC_AUTOMATIC_ACTION • OPC_OPERATOR_INIT_ACTION • OPC_TERMINAL_APPLICATION • OPC_NO_TERMINAL_ APPLICATION • OPC_GET_INSTRUCTION_TEXT
OPCDATA_ACTION_ANNOTATE	get/set	long		For actions of type OPC_AUTOMATIC_ACTION and OPC_OPERATOR_INIT_ACTION this attribute defines whether the HP Operations management server is required to create an annotation for the message that triggered the action. Possible values are: 0 (default): do not create an annotation 1: create an annotation.
OPCDATA_ACTION_ACK	get/set	long		Defines for actions of type OPC_AUTOMATIC_ACTION and OPC_OPERATOR_INIT_ACTION whether the HP Operations management server must acknowledge the message that triggered the action if the action was executed successfully. Possible values are: 0 (default): do not auto-acknowledge 1: auto-acknowledge.

Table 5-2 OPCDTYPE_ACTION_RESPONSE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_ACTION_RESULT	get/set	long		Result of the action: • OPC_ACTION_UNDEF • OPC_ACTION_DEF • OPC_ACTION_STARTED • OPC_ACTION_FINISHED • OPC_ACTION_FAILED
OPCDATA_ACTION_TIME	get/set	long		Time when the action was executed.
OPCDATA_NODENAME	get	str		Name of the node to perform action.
OPCDATA_ACTION_OUTPUT	get/set	str	L	Output of the executed action.

OPCDTYPE_ANNOTATION

Table 5-3 lists the attributes that are available for the Message Annotation data structure.

Table 5-3 OPCDTYPE_ANNOTATION

Attribute	Scope	Type	Properties	Description
OPCDATA_TIME	get	long		Time when the annotation was added.
OPCDATA_AUTHOR	get/set	str	L	Author of the annotation.
OPCDATA_ANNOTATION_TEXT OPCDATA_TEXT	get/set	str	L	Text of the annotation.
OPCDATA_ID	get/set	str		ID of the annotation.

OPCDTYPE_APPL_CONFIG

Table 5-4 lists the attributes that are available for the Application Configuration data structure.

Table 5-4 OPCDTYPE_APPL_CONFIG

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the application; this is the absolute path name from the hierarchy toplevel.
OPCDATA_LABEL	get/set	str	L	Symbol label displayed in the GUI
OPCDATA_CALL	get/set	str	L	Application, script, or program to be started on the managed node.
OPCDATA_PARAMETER	get/set	str	L	Parameter of the application call.
OPCDATA_USER	get/set	str		User name under which permissions the application will be started.
OPCDATA_PASSWORD	set	str		Password for the user.
OPCDATA_ID	get	str		ID of the application; this is the identifier of an application.
OPCDATA_DESCRIPTION	get/set	str	L	Short description of the application.
OPCDATA_SYMBOL	get/set	str		Symbol displayed in the GUI (APPLIC_TYPE).
OPCDATA_NODES	get/set	str		List of nodes where the application will be started; nodes names separated by pipes (|).
OPCDATA_GROUP_ID	get/set	str		Specifies the unique identifier of an HPOM application.
OPCDATA_TARGET	get/set	long		Specifies the method of the target machine selection. Possible Values are: • OPC_APPL_START_ON_TARGET_NODES • OPC_APPL_START_ON_SELECTED_NODES • OPC_APPL_START_ON_MGMT_SRV • OPC_APPL_START_ON_LOCAL_CLNT • OPC_APPL_START_ON_WWW_BROWSER

Table 5-4 OPCDTYPE_APPL_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_APP_TYPE	get/set	long		Specifies the application type. Possible values are: • OPC_APPL_INTERNAL • OPC_APPL_INTEGRATED
OPCDATA_APP_ACTION	get/set	long		This attribute is only valid for applications of type OPC_APPL_INTERNAL and specifies the environment in which the application will be executed. Possible values are: • OPC_APPL_INTERNAL_VIRTUAL • OPC_APPL_INTERNAL_PHYSICAL • OPC_APPL_INTERNAL_BROADCASt • OPC_APPL_INTERNAL_VIRTUAL_S
OPCDATA_APP_START_IN_TERM	get/set	long		This attribute is only valid for applications of type OPC_APPL_INTEGRATED and specifies the terminal type in which the application will be executed. Possible Values are: • OPC_APPL_NOTERM • OPC_APPL_TERMINAL • OPC_APPL_TERMOUT

Table 5-4 OPCDTYPE_APPL_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_PLTFRM_LOGINS	get/set	list		This attribute is only valid for applications of type OPC_APPL_INTERNAL and specifies a list of platform login configurations. Each configuration consists of the attributes: • OPCDATA_FAMILY_NAME Specifies the platform system family, e.g. UNIX. • OPCDATA_USER_NAME Specifies the user login for that platform family. • OPCDATA_PASSWORD Specifies the user password for that platform family.

OPCDTYPE_APPL_GROUP

Table 5-5 lists the attributes that are available for the Application Group Configuration data structure.

Table 5-5 OPCDTYPE_APPL_GROUP

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the application group; this is the absolute application group name from the toplevel of the hierarchy. The toplevel name can be NULL or "".
OPCDATA_LABEL	get/set	str	L	Specifies the label as it is printed in the GUI.
OPCDATA_ID	get	str		ID of the application group; this is the identifier of an application group.
OPCDATA_SYMBOL	get/set	str		Symbol displayed in the GUI (APPLIC_GROUP_TYPE).
OPCDATA_DESCRIPTION	get/set	str	L	Description for the application group.
OPCDATA_PARENT_ID	set	str		Specifies the unique identifier of the parent HPOM object.

OPCDTYPE_APPLIC

Table 5-6 lists the attributes that are available for the Application data structure.

Table 5-6 OPCDTYPE_APPLIC

Attribute	Scope	Type	Properties	Description
OPCDATA_APP_NAME OPCDATA_NAME	get/set	str		Name of the application, same as the label in the GUI.
OPCDATA_APP_GROUP OPCDATA_GROUP	get/set	str		Group name in which the application resides.
OPCDATA_APP_PARAMETER	get/set	str	L	Parameter for the application call.
OPCDATA_APP_USER	get/set	str		User name under which permissions the application will be started.
OPCDATA_APP_PASSWORD	get/set	str		Password for the user.

OPCDTYPE_APPLIC_RESPONSE

Table 5-7 lists the attributes that are available for the Application Response data structure.

Table 5-7 OPCDTYPE_APPLIC_RESPONSE

Attribute	Scope	Type	Properties	Description
OPCDATA_APP_IP_ADDRESS	get	long		IP address of the node.
OPCDATA_APP_STATUS	get	long		Status of the application: • OPC_ACTION_FINISHED • OPC_ACTION_FAILED
OPCDATA_APP_NODENAME	get	str		Name of the node where the application was started.
OPCDATA_APP_RESPONSE_ID	get	str		ID of the application response.
OPCDATA_APP_RESPONSE	get	str	L	Response of the application.

OPCDTYPE_CATEGORY_INFO

Table 5-1 lists the attributes that are available for the Category Info data structure.

Table 5-8 OPCDTYPE_CATEGORY_INFO

Attribute	Scope	Type	Properties			Description
OPCDATA_NAME	get/set	str				Name of the category.
OPCDATA_DESCRIPTION	get/set	str				Short description of the category.
OPCDATA_ID	get	str			L	ID of the category; this is the identifier of a category.

OPCDTYPE_CONFIG_EVENT

Table 5-9 lists the attributes that are available for the Configuration Event data structure.

Table 5-9 OPCDTYPE_CONFIG_EVENT

Attribute	Scope	Type	Properties	Description
OPCDATA_CONFIG_TASK	get	long		Configuration task. Used in base and extended format. Possible values are: • OPC_ADD_OPER • OPC_CHG_OPER • OPC_DEL_OPER • OPC_ASSIGN_OPER • OPC_DEASSIGN_OPER

Table 5-9 OPCDTYPE_CONFIG_EVENT (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_OBJECT_TYPE	get/set	str		Object type. Used in base and extended format. Possible values are: • OPC_MSGGRP_CHANGE • OPC_NODE_CHANGE • OPC_NODEGRP_CHANGE • OPC_APPLICATION_CHANGE • OPC_TEMPLATE_CHANGE • OPC_TEMPL_ASSIGNMENT_CHANGE • OPC_USER_CHANGE • OPC_PROFILE_CHANGE • OPC_TT_CHANGE • OPC_NOTIF_CHANGE • OPC_AUDIT_CHANGE • OPC_DB_MAINT_CHANGE • OPC_REGROUP_CHANGE • OPC_STAT_VAR_CHANGE • OPC_ECM_CFG_CHANGE • OPC_OUTAGE_CONF_CHANGE • OPC_TRANSACTION_CHANGE • OPC_APPLICATION_GROUP_CHANGE • OPC_NODEHIER_CHANGE • OPC_LAYOUT_GROUP_CHANGE • OPC_REGROUP_COND_CHANGE • OPC_TEMPLATE_GROUP_CHANGE • OPC_MASSIVE_CHANGE: sent by opccfgupld

Table 5-9 OPCDTYPE_CONFIG_EVENT (Continued)

Attribute	Scope	Type	Properties			Description
OPCDATA_OBJECT_TYPE	get/set	str				Object type. Used in base and extended format. Possible values are: • OPC_INSTR_IF_CHANGE • OPC_RESP_MAT_CHANGE: responsibility matrix changed • OPC_MGMTSV_CHANGE • OPC_RESP_MGRS_CHANGE • OPC_CCE_EVENT_ALL: all change events composed by |
OPCDATA_ID	get	str				Object ID. Used in base and extended format. Possible values are:
OPCDATA_GROUP_ID	get	str				Group ID. Used in extended format.
OPCDATA_PARENT_ID	get	str				Parent ID. Used in extended format.
OPCDATA_MSGGROUP_NAME	get	str				Message group name. Used in extended format.
OPCDATA_ASSIGN_OBJECT_TYPE	get	long				Object type of an assigned object. Used in extended format.
OPCDATA_STAT_VAR_CHANGE	get	long				Object type. Used for changing status variables by the opccfgout command.
OPCDATA_NAME	get	str				Name of status variable. Used for changing status variables by the opccfgout command.
OPCDATA_STAT_VAR_CURR	get	long				Status variable current value. Used for changing status variables by the opccfgout command.
OPCDATA_STAT_VAR_DEF	get	long				Status variable default value. Used for changing status variables by the opccfgout command.

OPCDTYPE_INFORM_USER

Table 5-10 lists the attributes that are available for the Inform User data structure.

Table 5-10 OPCDTYPE_INFORM_USER

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the HPOM user who will receive the information message.
OPCDATA_TEXT	get/set	str	L	Text of the message sent to the user.

OPCDTYPE_INSTR_IF

Table 5-11 lists the attributes that are available for the Instruction Text Interface data structure.

Table 5-11 OPCDTYPE_INSTR_IF

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the instruction text interface. The name must be unique.
OPCDATA_START_ON_MGMT_SV	get/set	long		If set to TRUE, the instruction text interface is called on the management server or on a controlled managed node.
OPCDATA_DESCRIPTION	get/set	str		Description of the instruction text interface.
OPCDATA_INSTR_CMD	get/set	str		Command that calls the instruction text interface.
OPCDATA_ID	get	str		UUID of the managed node where the instruction text interface is called.
OPCDATA_NODENAME	get/set	str		Hostname of the managed node. If OPCDATA_START_ON_MGMT_SV is set to TRUE, you do not need to set this attribute.
OPCDATA_IP_ADDRESS	get	str		IP address of the managed node.
OPCDATA_IP_FLAGS	get	long		IP flags of the managed node.
OPCDATA_NETWORK_TYPE	get	long		Network type of the managed node.
OPCDATA_USER	get/set	str		User account that calls the instruction text interface.
OPCDATA_RESOLVE_FOR_TTNS	get/set	long		If set to TRUE, the instruction text interface is called when forwarding messages to a trouble ticket system or a notification service. If set to FALSE, the instruction text interface is not called.
OPCDATA_OUTPUT_MODE	get/set	long		Output mode of the instruction text interface: • FALSE—Output only • TRUE—X application, for example

OPCDTYPE_LAYOUT_GROUP

Table 5-12 lists the attributes that are available for the Node Layout Group data structure.

Table 5-12 OPCDTYPE_LAYOUT_GROUP

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the HP Operations node layout group.
OPCDATA_LABEL	get/set	str	L	Specifies the layout group name shown in the GUI.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the HP Operations node layout group.
OPCDATA_SYMBOL	get/set	str		Symbol type of the HP Operations node layout group displayed in the GUI.
OPCDATA_PATH	get/set	str		Specifies the path from the Toplevel node layout group (Holding Area) to the node layout group. The path for Toplevel layout group is an empty string. Other layout groups are built by concatenation of upper layout groups and separating them with the a forward slash. For example, HoldingArea/LG1/LG1_1.
OPCDATA_SUBMAP_TITLE	get/set	str		Specifies the title of the submap.
OPCDATA_PARENT_ID	get/set	str		Specifies the unique ID of the parent layout group.
OPCDATA_ID	get	str		UUID of the HP Operations node layout group.

OPCDTYPE_MESSAGE

Table 5-13 lists the attributes that are available for the Message Attribute data structure.

Table 5-13 OPCDTYPE_MESSAGE

Attribute	Scope	Type	Properties	Description
OPCDATA_DATATYPE	get	long		Returns the type of the opcdata object.
OPCDATA_SEVERITY	get/set	long		Severity of the message. Possible values are: • OPC_SEV_UNCHANGED • OPC_SEV_UNKNOWN • OPC_SEV_NORMAL • OPC_SEV_WARNING • OPC_SEV_CRITICAL • OPC_SEV_MINOR • OPC_SEV_MAJOR
OPCDATA_CREATION_TIME	get/set	long		Time the message was created. The time is in UNIX format (seconds since Epoch). Default: the (local) time when the message was created.
OPCDATA_RECEIVE_TIME	get	long		Time the message was received by the management server.
OPCDATA_AACTION_ACK	get/set	long		Auto Acknowledge after successful execution of the Automatic Action 0 (default): do not auto-acknowledge 1: auto-acknowledge.
OPCDATA_ACTION_ANNOTATE	get/set	long		Defines whether HPOM creates start and end annotations for the automatic action. Possible values for the attribute are: 0 (default): do not create annotations 1: create annotations.

Table 5-13 OPCDTYPE_MESSAGE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_AACTION_STATUS	get/set	long		Status of the automatic action: • OPC_ACTION_UNDEF (default): no automatic action for this message defined. • OPC_ACTION_DEF: the automatic action for this message is defined but was not yet started. • OPC_ACTION_STARTED: the automatic action for this message is defined and was already started. • OPC_ACTION_FINISHED: the automatic action was started and completed successfully. • OPC_ACTION_FAILED: the automatic action was started and failed.
OPCDATA_OPACTION_ACK	get/set	long		Automatically acknowledge a message after a successful execution of the operator-initiated action. Possible values: 0 (default): do not auto-acknowledge 1: auto-acknowledge.
OPCDATA_OPACTION_ANNOTATE	get/set	long		Defines whether HPOM creates start and end annotations for the operator-initiated action. Possible values for the attribute are: 0 (default): do not create annotations 1: create annotations.
OPCDATA_OPACTION_STATUS	get	long		Status of the action: • OPC_ACTION_UNDEF • OPC_ACTION_DEF • OPC_ACTION_STARTED • OPC_ACTION_FINISHED • OPC_ACTION_FAILED
OPCDATA_NOTIFICATION	get/set	long		Notification.
OPCDATA_TROUBLETICKET	get/set	long		Forward message to Trouble Ticket system.

Table 5-13 OPCDTYPE_MESSAGE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_MSG_LOG_ONLY	get/set	long		Message is Server Log Only.
OPCDATA_TROUBLETICKET_ACK	get/set	long		Acknowledge message after forwarding it to the Trouble Ticket System.
OPCDATA_MSI_OUTPUT	get/set	long		The message will be forwarded to the MSI.
OPCDATA_INSTR_IF_TYPE	get/set	long		Type of the Instruction Interface: • OPC_INSTR_NOT_SET • OPC_FROM_OPC • OPC_FROM_OTHER • OPC_FROM_INTERNAL
OPCDATA_UNMATCHED	get	long		Defines whether or not the message matches a condition. Possible values are: 0 (default): the message was sent to the server because it matched a match condition 1: the message did not match a match condition of the assigned policies, but was forwarded nevertheless.
OPCDATA_TIME_ZONE_DIFF	get/set	long		Time difference to GMT in seconds.
OPCDATA_FORWARDED_FROM	get	long		This flag signals whether the message is forwarded from another management server.
OPCDATA_IS_READONLY	get	long		The message is read only (TRUE) or can be acknowledged on this server.

Table 5-13 OPCDTYPE_MESSAGE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_MSGSRC_TYPE	get	long		Defines the Message Source Type, the source which originated this message, for example, the monitor agent. Possible values are: • OPC_OPCMSG_SRC: opcmsg(1|3) policy • OPC_LOGFILE_SRC: logfile • OPC_MONITOR_SRC: monitor agent • OPC_SNMPTRAP_SRC: SNMP trap interceptor • OPC_SVMSI_SRC: server MSI • OPC_AGTMSI_SRC: agent MSI • OPC_LEGLINK_SRC: legacy link interface • OPC_SCHEDULE_SRC: scheduler
OPCDATA_TIME_OWNED	get	long		Time an operator took ownership of a message.
OPCDATA_DATA_INFO	get/set	long		Additional information about the message: • OPC_REMARK_FOR_ACK
OPCDATA_MSG_STATUS	get	long		Status of the message: • OPC_MSG_ACTIVE • OPC_MSG_HISTORY
OPCDATA_ACKNOWLEDGE_TIME	get	long		Time when the message was acknowledged.
OPCDATA_NUM_ANNOTATIONS	get	long		Number of annotations.
OPCDATA_APPLICATION	get/set	str	L	Application which produced the message. Default: empty string.
OPCDATA_GROUP	get/set	str		Message group. Default: empty string.
OPCDATA_MSGTEXT	get/set	str	L	Message Text. Default: empty string.

Table 5-13 OPCDTYPE_MESSAGE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_ORIGMSGTEXT	get/set	str	L	Original Message Text. Allows you to set additional source information for a message. It is only useful if the message text was reformatted but the HPOM operator needs to have access to the original text as it appeared before formatting. Default: empty string.
OPCDATA_MSGTYPE	get/set	str		Message Type. This attribute is used to group messages into subgroups, e.g., to denote the occurrence of a specific problem. This information may be used by event correlation engines. Default: empty string.
OPCDATA_NODENAME	get/set	str		Name of the node producing the message. The message is only handled by the HPOM manager if this system is part of the HPOM Node Bank. Default: local node name.
OPCDATA_OBJECT	get/set	str	L	Object name to use for the HPOM message. Default: empty string.
OPCDATA_MSGSRC	get	str		Message source. For example, the name of the encapsulated logfile if the message originated from logfile encapsulation or the interface name if the message was sent via an instance of the Message Stream Interface. Default: empty string.
OPCDATA_MSGID	get	str		The unique ID of the message.
OPCDATA_AACTION_NODE	get/set	str		Defines the node on which the automatic action should run. Default: value of OPCDATA_NODENAME.
OPCDATA_AACTION_CALL	get/set	str	L	Command to use as automatic action for the HPOM message. Default: empty string.
OPCDATA_OPACTION_NODE	get/set	str		Defines the node on which the operator-initiated action should run. Default: value of OPCDATA_NODENAME.
OPCDATA_OPACTION_CALL	get/set	str	L	Command to use as operator-initiated action for the HPOM message. Default: empty string. Call of the operator-initiated action.

Table 5-13 OPCDTYPE_MESSAGE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_INSTR_IF	get/set	str		Name of the external instruction text interface. The external instruction text interface must be configured in HPOM. Default: empty string.
OPCDATA_INSTR_PAR	get/set	str	L	Parameters for a call to the external instruction text interface. Default: empty string.
OPCDATA_OWNED_BY	get	str		Name of the operator who owns the message.
OPCDATA_OPTION_VAR	set	str	L	A string containing the optional parameters used for resolving the $OPTION variables by the message interceptor. The string should have the format [<var>=<value>]* with <var> and <value> not containing spaces or the '=' character.
OPCDATA_ACKNOWLEDGE_OP	get	str		Name of operator who has acknowledged the message.
OPCDATA_MSG_KEY	get/set	str	L	Additional message attribute for customized message handling.
OPCDATA_SERVICE_NAME	get/set	str		Specifies the service name.
OPCDATA_ORIGMSGID	get	str		Unique identifier of the original message. This is set when the message ID was changed because of a message change.
OPCDATA_NUM_DUPLICATES	get	long		Number of duplicate messages of this message. This field is always zero (0) for messages passing through the Message Stream Interface (MSI) because the duplicate count is increased for the original message only and not for new messages arriving in the MSI.
OPCDATA_LAST_REC_TIME	get	long		Contains the time when the last duplicate message was received.
OPCDATA_MSG_KEY_RELATION	get/set	str	L	Specifies the message key relation. Can contain patterns.
OPCDATA_MSG_KEY_RELATION_ICASE	get/set	long		Case sensitivity of message key relation: 0=case-sensitive, !=0 not case-sensitive.

Table 5-13 OPCDTYPE_MESSAGE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_MSG_KEY_RELATION_SEPS	get/set	str		Field separators for message key relations.
OPCDATA_MSG_GEN_NODENAME	get	string		Name of the node where the event occurred.
OPCDATA_MSG_GEN_IP_ADDRESS	get	long		IP address of the node where the event occurred.
OPCDATA_MSG_GEN_NETWORK_TYPE	get	long		Network type of the node where the event occurred.

OPCDTYPE_MESSAGE_EVENT

Table 5-14 lists the attributes that are available for the Message Event data structure.

Table 5-14 OPCDTYPE_MESSAGE_EVENT

Attribute	Scope	Type	Properties	Description
OPCDATA_EVENT_FLAG	get	long		Type of event that occurred: • OPC_MSG_EVENT_ACK • OPC_MSG_EVENT_UNACK • OPC_MSG_EVENT_OWN • OPC_MSG_EVENT_DISOWN • OPC_MSG_EVENT_ANNO • OPC_MSG_EVENT_NO_ANNO • OPC_MSG_EVENT_AA_START • OPC_MSG_EVENT_AA_END • OPC_MSG_EVENT_OA_START • OPC_MSG_EVENT_OA_END • OPC_MSG_EVENT_BUFFER • OPC_MSG_EVENT_UNBUFFER • OPC_MSG_EVENT_MODIFY • OPC_MSG_EVENT_HIGHLIGHT • OPC_MSG_EVENT_CHGSEV • OPC_MSG_EVENT_MSGSHG • OPC_MSG_EVENT_CMA_UPDATE • OPC_MSG_EVENT_DEL: Message was deleted.

Table 5-14 OPCDTYPE_MESSAGE_EVENT (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_EVENT_FLAG	get	long		Type of event that occurred: • OPC_MSG_EVENT_ACTIVE_RECEIVED_MSG[a]: New active message received. In case you registered for this event and if OPCUIWWW_NEW_MSG_NO_DB is TRUE, the new message is received instead of an event. • OPC_MSG_EVENT_DUPL_RECEIVED[a]: Duplicate of message received. • OPC_MSG_EVENT_ANNO_ADD[a]: Annotation added. • OPC_MSG_EVENT_ANNO_SET[a]: Annotation modified. • OPC_MSG_EVENT_ANNO_DEL[a]: Annotation deleted. • OPC_MSG_EVENT_READONLY_MSG[a]: Message became read-only. • OPC_MSG_EVENT_CTRL_MSG[a]: Message became a control switch message. • OPC_MSG_EVENT_ACTIVE_RECEIVED[a]: New active message received. • OPC_MSG_EVENT_HISTORY_RECEIVED[a]: New history message received.
OPCDATA_MSGID	get	str		ID of the message.
OPCDATA_ANNOTATION_ID	get	str		ID of the message annotation.
OPCDATA_RECEIVE_TIME	get	long		Specifies the receiving time of the message event in UNIX format.
OPCDATA_USER_NAME	get	str		Specifies the name of the target user in a highlight event.
OPCDATA_SEVERITY	get	long		Contains the current severity of the message.

Table 5-14 OPCDTYPE_MESSAGE_EVENT (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_OLD_SEVERITY	get	long		Contains the previous severity in case of an OPC_MSG_EVENT_MODIFY event.
OPCDATA_SERVICE_NAME	get	str		Specifies the service name.

a. In HPOM 8.xx, this event occurrs *only* if you open Message Event Interface (MEI) with OPCSVIF_MSG_EVENTS_ALL.

OPCDTYPE_MESSAGE_GROUP

Table 5-15 lists the attributes that are available for the Message Group data structure.

Table 5-15 OPCDTYPE_MESSAGE_GROUP

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the HPOM message group.
OPCDATA_LABEL	get/set	str	L	Specifies the label shown in the GUI.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the HPOM message group.
OPCDATA_SYMBOL	get/set	str		Symbol type displayed in the GUI.

OPCDTYPE_MESSAGE_ID

Table 5-16 lists the attributes that are available for the Message ID data structure.

Table 5-16 OPCDTYPE_MESSAGE_ID

Attribute	Scope	Type	Properties	Description
OPCDATA_MSGID OPCDATA_ID	get/set	str		Unique identifier of a message (Message ID).

OPCDTYPE_MONITOR_MESSAGE

Table 5-17 lists the attributes that are available for the Monitor Message data structure.

Table 5-17 OPCDTYPE_MONITOR_MESSAGE

Attribute	Scope	Type	Properties	Description
OPCDATA_MON_VAR	set	str		Name of the monitored object.
OPCDATA_MON_VALUE	set	double		Monitor value.
OPCDATA_OPTION_VAR	set	str	L	A string containing the optional parameters used for resolving the $OPTION variables by the monitor agent. The string should have the format [<var>=<value>]* with <var> and <value> not containing spaces or the '=', '(' or ')' characters.
OPCDATA_OBJECT	set	str	L	Message object.

OPCDTYPE_NODE

Table 5-18 lists the attributes that are available for the Managed Node data structure.

Table 5-18 OPCDTYPE_NODE

Attribute	Scope	Type	Properties	Description
OPCDATA_STATUS	get/set	long		Status of a previous assignment or deassignment; either OPC_ERR_OK or appropriate error code.
OPCDATA_NETWORK_TYPE	get/set	long		Type of the network: • OPC_NETWORK_NO_NODE • OPC_NETWORK_IP • OPC_NETWORK_OTHER • OPC_NETWORK_UNKNOWN
OPCDATA_MACHINE_TYPE	get/set	long		See OPCDATA_MACHINE_TYPE of “OPCDTYPE_NODE_CONFIG” on page 576.
OPCDATA_IP_ADDRESS	get/set	long		IP address of the node.
OPCDATA_NODENAME, OPCDATA_NAME	get/set	str		Name of the node.
OPCDATA_LABEL	get/set	str	L	Specifies the label of the node shown in the GUI.
OPCDATA_NODE_TYPE, OPCDATA_CONTROL	get/set	long		Possible values are: • OPC_NODE_DISABLED • OPC_NODE_CONTROLLED • OPC_NODE_MONITORED • OPC_NODE_MESSAGE_ALLOWED

Table 5-18 OPCDTYPE_NODE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_TERMINAL	get/set	long		Specifies the terminal type for that node. Possible values are: • OPC_TERM_HPTERM • OPC_TERM_XTERM • OPC_TERM_DTTERM • OPC_TERM_NONE_TERM
OPCDATA_SYMBOL	get/set	str		Specifies the symbol shown in the GUI.
OPCDATA_ID	get/set	str		Specifies the unique ID of the node. The ID has a higher priority than the name when this structure is used to specify a node for access.
OPCDATA_LAYOUTGRP_ID	get/set	str		Specifies the layout group, necessary for some functions.

OPCDTYPE_NODE_CONFIG

Table 5-19 lists the attributes that are available for the Managed Node Configuration data structure.

Table 5-19 OPCDTYPE_NODE_CONFIG

Attribute	Scope	Type	Properties	Description
OPCDATA_STATUS	get/set	long		Status of a previous assignment or deassignment; either OPC_ERR_OK or appropriate error code.
OPCDATA_NAME, OPCDATA_NODENAME	get/set	str		Name of the HP Operations node.
OPCDATA_LABEL	get/set	str	L	Label of the HP Operations node.
OPCDATA_IP_ADDRESS	get	long		IP address of the HP Operations node.
OPCDATA_SYMBOL	get/set	str		Symbol type of the HP Operations node displayed in the GUI.
OPCDATA_NETWORK_TYPE	get/set	long		Type of the network: • OPC_NETWORK_NO_NODE • OPC_NETWORK_IP • OPC_NETWORK_OTHER • OPC_NETWORK_UNKNOWN • OPC_NODE_PATTERN_IP_ADDR • OPC_NODE_PATTERN_IP_NAME • OPC_NODE_PATTERN_OTHER: deprecated (mapped internally on OPC_NODE_PATTERN_IP_NAME since HPOM 9.00)
OPCDATA_CONTROL	get/set	long		Control type of the system: • OPC_NODE_CONTROLLED • OPC_NODE_MONITORED • OPC_NODE_MESSAGE_ALLOWED • OPC_NODE_DISABLED

Table 5-19 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_HEARTBEAT_POLL_INTERVAL	get/set	str		Duration of the heartbeat-polling interval.
OPCDATA_HEARTBEAT_POLL_TYPE	get/set	long		Type of heartbeat polling used for the node: • OPC_HEARTBEAT_NONE • OPC_HEARTBEAT_RPC_ONLY • OPC_HEARTBEAT_FROM_AGENT • OPC_HEARTBEAT_RPC_AND_PING
OPCDATA_HEARTBEAT_POLL_ENABLE	get/set	long		Heartbeat polling on or off (TRUE/FALSE).
OPCDATA_AUTO_INSTALL	get/set	long		Automatic installation/deinstallation on or off (TRUE/FALSE).
OPCDATA_AUTO_UPDATE	get/set	long		Automatic update of system resource files (TRUE/FALSE).
OPCDATA_INSTALL_USER	get/set	str		User name under which the automatic installation will be done.
OPCDATA_TERMINAL	get/set	long		Virtual terminal: • OPC_TERM_DTTERM • OPC_TERM_HPTERM • OPC_TERM_XTERM
OPCDATA_CONSOLE_COMMAND	get/set	str		Command for the physical console.
OPCDATA_CONSOLE_OPT1	get/set	str		Options for the physical console command.
OPCDATA_CONSOLE_OPT2	get/set	str		Options for the physical console command.
OPCDATA_CONSOLE_OPT3	get/set	str		Options for the physical console command.
OPCDATA_VTERM_FONT	get/set	str		Font for the virtual terminal.
OPCDATA_MSI_ENABLE_OUTPUT	get/set	long		Enable output to the Message Stream Interface (MSI).
OPCDATA_MSI_ALLOW_AACTIONS	get/set	long		Allow externally defined automatic actions.

Table 5-19 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_MSI_ALLOW_OPACTIONS	get/set	long		Allow externally defined operator-initiated actions.
OPCDATA_INSTALL_METHOD	get/set	long		Installation method: • OPC_INSTALL_OpC • OPC_INSTALL_OpC_ASYNC • OPC_INSTALL_SD • OPC_INSTALL_SD_ASYNC
OPCDATA_COMM_TYPE	get/set	long		Specifies the communication type of the node. Not all nodes support all communication types. Possible values are: • OPC_COMM_OPC_INTERFACE • OPC_COMM_BBC
OPCDATA_ID	get/set	str		Specifies the unique ID of the node. The ID has a higher priority than the name when this structure is used to specify a node for access.
OPCDATA_BUFLIMIT_ENABLE	get/set	long		Enables (val != 0) or disables (val = 0) the buffer file size limitation on the node.
OPCDATA_BUFLIMIT_SIZE	get/set	long		Specifies the maximum size of the buffer file in kBytes (n*1024 Bytes), if buffer file size limitation is enabled. The minimum is 1.000 kBytes. If it is set to less than 1.000 kBytes, it will be set to 1.000 kBytes. The default is 10.000 kBytes.

Table 5-19 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_BUFLIMIT_SEVERITY	get/set	long		Specifies the severity of messages that will stay in the buffer file when the limit has been reached. Messages with a severity lower than OPCDATA_BUFLIMIT_SEVERITY are discarded. Possible values are: • OPC_SEV_UNKNOWN • OPC_SEV_NORMAL • OPC_SEV_WARNING • OPC_SEV_CRITICAL • OPC_SEV_MINOR • OPC_SEV_MAJOR

OPCDTYPE_NODE_GROUP

Table 5-20 lists the attributes that are available for the Node Group data structure.

Table 5-20 OPCDTYPE_NODE_GROUP

Attribute	Scope	Type	Properties	Description
OPCDATA_STATUS	get/set	long		Status of a previous assignment or deassignment; either OPC_ERR_OK or appropriate error code.
OPCDATA_NAME	get/set	str		Name of the HP Operations node group.
OPCDATA_LABEL	get/set	str	L	Label of the HP Operations node group; displayed in the GUI.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the HP Operations node group.
OPCDATA_SYMBOL	get/set	str		Symbol type displayed in the GUI.
OPCDATA_ID	get	str		ID of the node group.

OPCDTYPE_NODEHIER

Table 5-21 lists the attributes that are available for the Node Hierarchy data structure.

Table 5-21 OPCDTYPE_NODEHIER

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the HP Operations node hierarchy.
OPCDATA_LABEL	get/set	str	L	Specifies the node hierarchy name shown in the GUI.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the HP Operations node hierarchy.
OPCDATA_ID	get	str		UUID of the HP Operations node hierarchy.
OPCDATA_SYMBOL	get/set	str		Symbol type of the HP Operations node hierarchy displayed in the GUI.
OPCDATA_HOLDING_AREA_ID	get/set	str		Specifies the UUID of the holding area. This is the OPC_EMPTY_UUID or the UUID of a valid layout group.

OPCDTYPE_NOTI_SCHEDULE

Table 5-23 lists the attributes that are available for the Notification Schedule data structure.

Table 5-22 OPCDTYPE_NOTI_SCHEDULE

Attribute	Scope	Type	Properties	Description
OPCDATA_NOTI_SCHEDULE_DAY	get/set	long		Day of the notification schedule. Possible values are: Sunday OPCDATA_NOTI_SCHEDULE_SUNDAY = 0 Monday OPCDATA_NOTI_SCHEDULE_MONDAY = 1 Tuesday OPCDATA_NOTI_SCHEDULE_TUESDAY = 2 Wednesday OPCDATA_NOTI_SCHEDULE_WEDNESDAY = 3 Thursday OPCDATA_NOTI_SCHEDULE_THURSDAY = 4 Friday OPCDATA_NOTI_SCHEDULE_FRIDAY = 5 Saturday OPCDATA_NOTI_SCHEDULE_SATURDAY = 6
OPCDATA_NOTI_SCHEDULE_START_TIME	get/set	str		Start time of the notification schedule in seconds (format 0-86400).
OPCDATA_NOTI_SCHEDULE_END_TIME	get/set	str		End time of the notification schedule in seconds (format 0-86400).
OPCDATA_ID	get/set	str		UUID of the notification service scheduled by this data structure.

OPCDTYPE_NOTI_SERVICE

Table 5-23 lists the attributes that are available for the Notification Service data structure.

Table 5-23 OPCDTYPE_NOTI_SERVICE

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the notification service. The name must be unique.
OPCDATA_NOTI_SERVICE_CALL	get/set	str		Call or command of the notification service.
OPCDATA_ID	get	str		UUID of the notification service.

OPCDTYPE_REGROUP_COND

Table 5-24 lists the attributes that are available for the Regroup Condition data structure.

Table 5-24 OPCDTYPE_REGROUP_COND

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the message regroup condition.
OPCDATA_ID	get	str		ID of the regroup condition. This is the identifier of a condition.
OPCDATA_SEVERITY	get/set	long		Severity level of the regroup condition.
OPCDATA_APPLICATION	get/set	str	L	List of applications, separated by a "|", that generated the message.
OPCDATA_OBJECT	get/set	str	L	List of object names, separated by a "|".
OPCDATA_MESSAGE_GROUP	get/set	str		List of message group names. Each name is separated by a "|".
OPCDATA_NODE	get/set	str		List of nodes, separated by a "|".
OPCDATA_MESSAGE_TEXT	get/set	str	L	Search pattern.
OPCDATA_FIELD_SEPARATORS	get/set	str		Field-separating characters.
OPCDATA_CASE_SENSITIVE_CHECK	get/set	long		Check if case sensitive or not.
OPCDATA_NEW_MESSAGE_GROUP	get/set	str		Name of the new message group.
OPCDATA_SERVICE_NAME	get/set	str		Specifies the service name.

OPCDTYPE_TEMPLATE_INFO

Table 5-25 lists the attributes that are available for the Policy data structure.

Table 5-25 OPCDTYPE_TEMPLATE_INFO

Attribute	Scope	Type	Properties	Description
OPCDATA_STATUS	get/set	long		Status of a previous assignment or deassignment; either OPC_ERR_OK or appropriate error code.
OPCDATA_NAME	get/set	str		Name of the policy or policy group.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the policy or policy group.
OPCDATA_TYPE	get/set	long		Type of the policy or policy group: • OPC_UNKNOWN_TEMPLATE • OPC_CONSOLE_TEMPLATE • OPC_OPCMSG_TEMPLATE • OPC_LOGFILE_TEMPLATE • OPC_MONITOR_TEMPLATE • OPC_SNMP_TEMPLATE • OPC_EC_TEMPLATE • OPC_SCHEDULE_TEMPLATE • OPC_TEMPLATE_GROUP
OPCDATA_ID	get	str		UUID of the policy or policy group. This ID is generated when the policy is created.

OPCDTYPE_USER_CONFIG

Table 5-26 lists the attributes that are available for the User Configuration data structure.

Table 5-26 OPCDTYPE_USER_CONFIG

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Login name of the HPOM user.
OPCDATA_LABEL	get/set	str	L	Specifies the user name shown in the GUI.
OPCDATA_PASSWORD	set	str		Password of the HPOM user; necessary for login of the HPOM user.
OPCDATA_REAL_NAME	get/set	str	L	Real name of the HPOM user.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the HPOM user.
OPCDATA_SYMBOL	get/set	str		Symbol type displayed in the GUI.
OPCDATA_ID	get	str		UUID of the HPOM user. This parameter is set when the user is created.
OPCDATA_USER_ROLE	get/set	long		Role of the HPOM user. The user can be of the type: • OPC_ROLE_OPERATOR • OPC_ROLE_VIRTUAL_OPERATOR • OPC_ROLE_TEMPLATE_ADMIN • OPC_ROLE_PROFILE
OPCDATA_PERFORM_ACTION	get/set	long		Specifies whether the HPOM user is allowed to perform operator-initiated actions.
OPCDATA_ACKNOWLEDGE	get/set	long		Specifies whether the HPOM user is allowed to acknowledge messages.
OPCDATA_OWN_FLAG	get/set	long		If set TRUE, user can own messages.
OPCDATA_CHANGE_MSG_ATTR	get/set	long		If set TRUE, user can change message attributes.

OPCDTYPE_USER_RESP_ENTRY

Table 5-27 lists the attributes that are available for the User Responsibility Entry data structure.

Table 5-27 OPCDTYPE_USER_RESP_ENTR

Attribute	Scope	Type	Properties	Description
OPCDATA_NODEGROUP_ID	get/set	str		Node group ID—this is the identifier of the HP Operations node group. If the NODEGROUP_NAME is also given, the NODEGROUP_ID overrides this information.
OPCDATA_NODEGROUP_NAME	get/set	str		Name of the node group. This is an alternative identifier of the HP Operations node group. If the NODEGROUP_ID is also given, the NODEGROUP_ID overrides this information.
OPCDATA_MSGGROUP_NAME	get/set	str		Name of the message group. This is the identifier of an HPOM message group.

OPCREGCOND

HPOM provides a user-accessible data type to define registration conditions as the mechanism to register with the HPOM Interfaces.

Table 5-28 lists the registration conditions of the function `opcif_register()`, see also “opcif_register()” on page 114 for more information.

Table 5-28 opcregcond

Attribute	Scope	Type	Properties	Description
OPCREG_APPLICATION	get/set	str		Registers for the message attribute application.
OPCREG_APP_RESPONSE_ID	get/set	str		Registers for application responses with the ID=OPCREG_APP_RESPONSE_ID.
OPCREG_GROUP	get/set	str		Registers for the message attribute message group.
OPCREG_MSG_EVENT_MASK	get/set	long		Registers for events matching OPCREG_MSG_EVENT_MASK.
OPCREG_MSGTYPE	get/set	str		Registers for the message attribute message type.
OPCREG_NODENAME	get/set	str		Registers for the message attribute node.
OPCREG_OBJECT	get/set	str		Registers for the message attribute object.
OPCREG_OPERATOR	get/set	str		Registers for the message events of certain operators.
OPCREG_SEVERITY	get/set	long		Registers for the message attribute severity.

6 Service Navigator Interfaces and APIs

In this Chapter

The following integration facilities are provided for Service Navigator integrations:

- ❏ XML Data Interface
- ❏ C++ APIs of the service engine
  - The Service Operations Interface Classes
  - The Registration Interface Classes

  These APIs are C++ interfaces and come complete with:
  - `opcsvcapi.h` header file
  - `libopcsvcapi.sl` shared library

NOTE By default, remote access to the service engine is disabled. See “Allowing Remote Access to the Service Engine” on page 636 for information on how to allow remote access to the service engine.

The XML Data Interface

The XML Data Interface allows you to write or get service configuration directly into or from the service engine via a filesystem socket. The XML data interface:

- Allows you to *write* the service configuration directly into the service engine. The configuration syntax follows the XML rules defined in the document type definition (DTD) `operations.dtd`.
- Allows you to *get* the current service configuration and service status directly from the service engine. The output syntax follows the XML rules defined in the DTD `results.dtd`.

You can test your XML commands interactively using the `opcsvcterm` program. This is an interface to the service engine that inputs XML into stdin and outputs XML to stdout. See also the man page *opcsvcterm(1M)* for more information.

The filesystem socket is placed in:

`/var/opt/OV/sockets/OpC/opcsvcm`

XML Notation Used

The format of the operations and results files is based on the World Wide Web Consortium Extended Markup Language (XML). The DTDs for the Service Navigator XML syntax are printed in this section and are also available on the HP Operations management server as:

```
/etc/opt/OV/share/conf/OpC/mgmt_sv/dtds/services.dtd
```

```
/etc/opt/OV/share/conf/OpC/mgmt_sv/dtds/operations.dtd
```

```
/etc/opt/OV/share/conf/OpC/mgmt_sv/dtds/loggings.dtd
```

```
/etc/opt/OV/share/conf/OpC/mgmt_sv/dtds/results.dtd
```

All DTDs are also available in XML Schema Definition (XSD) format in the same directory. This alternative format is based on XML and therefore easier to read with XML editors.

The following syntax rules apply:

- **Case Sensitive**

  The Service Navigator XML parser is case sensitive; the XML tags must be specified as defined in the DTD.

- **XML Processing Instruction**

  Each XML file must *start* with an XML processing instruction:

  ```
  <?xml version="1.0" ?>
  ```

  Comments or any other tag are not allowed before this instruction.

❑ **Codeset**

If no codeset is defined, the default value UTF-8 is used.

```
<?xml version="1.0" encoding="UTF-8"?>
```

The following codesets can be used with Service Navigator:

Table 6-1 Supported Codesets

Language	Codeset	
Czech	UTF-8 UTF-16BE UTF-16LE	ISO-8859-2
Japanese		Shift_JIS
Korean		EUC-KR
Russian		ISO-8859-5
Simplified Chinese		GB2312
Traditional Chinese		Big5
Western European (for example, English, French, German, or Spanish)		ISO-8859-1 ISO-8859-15 roman8

The codesets are the standard codeset names as defined by IANA (Internet Assigned Numbers Authority).

- **Namespaces**

  The following namespaces are used by the Service Navigator service DTDs:

  - XML namespace of the `service.dtd`:

    `http://www.hp.com/OV/opcsvc`

  - XML namespace for the `operations.dtd`:

    `http://www.hp.com/OV/opcsvcoperations`

  - XML namespace for the `loggings.dtd`:

    `http://www.hp.com/OV/opcsvcloggings`

  - XML namespace for the `results.dtd`:

    `http://www.hp.com/OV/opcsvcresults`

  Name spaces are specified within the toplevel XML tag and are used to uniquely identify the XML tags. For example, a file `services.xml` should start like this:

```
<?xml version='1.0' ?>
<Services xmlns="http://www.hp.com/OV/opcsvc"
version="1.0">
```

- **Comments**

  XML provides a mechanism for commenting code which has the same syntax as for HTML comments: `<!-- comment -->`; comments must not occur within declarations or inside element tags.

- **Content Model Operators**

  The following content model operators occur in the DTD:

Table 6-2 XML Content Model Operators

Symbol	Usage
’	Strict ordering
|	Selection
+	Repetition; minimum one
*	Repetition
?	Optional
()	Grouping

- **#PCDATA**

  Elements that have character content are declared as `#PCDATA`.

- **EMPTY**

  Elements that are empty are declared as `EMPTY`.

Note that the term “objects” used in the following tables refers to the following configuration objects:

- Services
- Actions
- Operators
- Loggings
- Propagation rules
- Calculation rules

The Operations Tags

The following is the Document Type Definition (DTD) for the operations file. Note that the `operations.dtd` includes the `service.dtd` which is described in more detail in the *Service Navigator Concepts and Configuration Guide*.

```
<!-- XML DTD for service engine operations  -->
<!-- Operations is the root element         -->

<!ENTITY % ServicesDTD SYSTEM "service.dtd">
%ServicesDTD;
<!-- Operations into ServiceEngine -->

<!ELEMENT Operations ((Add |
                       Replace |
                       Remove |
                       SetLabel |
                       SetAttributes |
                       RemoveAttributes |
                       RemovePrefAttributes |
                       List |
                       ListAssignments |
                       AssignServices |
                       DeassignServices |
                       GetElementStatus |
                       GetAssocStatus |
                       GetElementMultiStatus |
                       GetAssocMultiStatus |
                       GetRootCauses |
                       GetImpacts |
                       Registration |
                       AddLoggings |
                       RemoveLoggings |
```

```
                        ListLoggings |
                        GetServicesForMessage |
                        GetViewsForOriginalId |
                        Dump |
                        GetStatusCalculations |
                        GetDefaultStatusCalculation
                        )*)>
<!ATTLIST Operations xmlns   CDATA #IMPLIED
              version CDATA #IMPLIED>

<!ELEMENT Add                ((Services | ServicesRef),
                             ((ServiceRefs?, ActionRefs?,
                             PropRuleRefs?,
                               CalcRuleRefs?, OperatorRefs?)
                              | All))>
<!ELEMENT Replace            ((Services | ServicesRef),
                             ((ServiceRefs?, ActionRefs?,
                             PropRuleRefs?,
                               CalcRuleRefs?, OperatorRefs?)
                              | All))>

<!ELEMENT Remove             ((ServiceRefs?, ActionRefs?,
                             PropRuleRefs?,
                               CalcRuleRefs?, OperatorRefs?)
                             | All)>

<!ELEMENT ServiceRefs        (ServiceRef* | All)>
<!ELEMENT ActionRefs         (ActionRef* | All)>
<!ELEMENT PropRuleRefs       (PropRuleRef* | All)>
<!ELEMENT CalcRuleRefs       (CalcRuleRef* | All)>
<!ELEMENT OperatorRefs       (OperatorRef* | All)>

<!ELEMENT SetLabel           (ServiceRef, Label)>
```

```
<!ELEMENT SetAttributes       (ServiceRef, Attribute*)>
<!ELEMENT RemoveAttributes    (ServiceRef, Name*)>
<!ELEMENT RemovePrefAttributes (ServiceRef,Name?)>

<!-- empty tags means all of the sort, a name means only that -->
<!ELEMENT List                (Recursive?,(Full | Depth)?,
                               ((ServiceRefs?, ActionRefs?,
                                 PropRuleRefs?,CalcRuleRefs?)
                                | All | OperatorRefs))>

<!ELEMENT ListAssignments     (ServiceRef+)>

<!ELEMENT AssignServices      (OperatorRef,ServiceRef+)>
<!ELEMENT DeassignServices    (OperatorRef,(ServiceRef+ | All))>

<!ELEMENT GetElementStatus    (ServiceRef | OperatorRef)>
<!ELEMENT GetAssocStatus      ((Dependency | Composition |
                              OperatorAssignment),
                               SourceRef, TargetRef)>
<!ELEMENT GetElementMultiStatus    ((ServiceRef | OperatorRef),
                                   CalculationId?)>
<!ELEMENT GetAssocMultiStatus      ((Dependency | Composition |
                                   OperatorAssignment),
                                     SourceRef, TargetRef,
                                   CalculationId?)>

<!ELEMENT GetRootCauses       ((ServiceRef | OperatorRef),
                              CalculationId?)>
<!ELEMENT GetImpacts          (ServiceRef, NoCause?,
                             CalculationId?>

<!ELEMENT AddLoggings         (RegCondition+)>
<!ELEMENT RemoveLoggings      (ServiceRef*)>
```

```
<!ELEMENT ListLoggings        (ServiceRef*)>

<!ELEMENT Registration        ((RegCondition*|All),
                              WithLabelChanges?,
                              WithAttributeChanges?,
                              WithMultiStatusChanges?)>
<!-- recursive optionally until level -->
<!ELEMENT RegCondition        ((ServiceRef | OperatorRef),
                               (Depth | Recursive)?)>

<!ELEMENT GetViewsForOriginalId (OriginalIdRef, OperatorRef)>
<!ELEMENT GetServicesForMessage (#PCDATA) >

<!ELEMENT Dump                (#PCDATA)>

<!ELEMENT ServicesRef      (#PCDATA)>
<!ELEMENT OperatorRef      (#PCDATA)>
<!ELEMENT OriginalIdRef       (#PCDATA)>

<!ELEMENT GetStatusCalculations    EMPTY>
<!ELEMENT GetDefaultStatusCalculationEMPTY>

<!ELEMENT All              EMPTY>
<!ELEMENT Full             EMPTY>
<!ELEMENT Recursive        EMPTY>

<!ELEMENT WithLabelChanges           EMPTY>
<!ELEMENT WithAttributeChanges   EMPTY>
<!ELEMENT WithMultiStatusChanges EMPTY>
<!ELEMENT NoCause                EMPTY>
<!-- EOF -->
```

Table 6-3 The Operations Tags

Tag	Required?	Description
<Operations>	Required.	Is the root element. It contains the following tags: • <Add> • <Replace> • <Remove> • <SetLabel> • <SetAttributes> • <RemoveAttributes> • <RemovePrefAttributes> • <List> • <ListAssignments> • <AssignServices> • <DeassignServices> • <GetElementStatus> • <GetAssocStatus> • <GetElementMultiStatus> • <GetAssocMultiStatus> • <GetRootCauses> • <GetImpacts> • <Registration> • <AddLoggings> • <RemoveLoggings> • <ListLoggings> • <GetServicesForMessage> • <GetViewsForOriginalId> • <Dump> • <GetStatusCalculations> • <GetDefaultStatusCalculation>

Table 6-3 The Operations Tags (Continued)

Tag	Required?	Description
<Add>	Any number possible.	Adds objects to the configuration. It contains the following tags: • <Services> • <ServicesRef> • <ServiceRefs> • <ActionRefs> • <PropRuleRefs> • <CalcRuleRefs> • <OperatorRefs> • <All>
<Replace>	Any number possible.	Replaces objects in the configuration. It contains the following tags: • <Services> • <ServicesRef> • <ServiceRefs> • <ActionRefs> • <PropRuleRefs> • <CalcRuleRefs> • <OperatorRefs> • <All>
<Remove>	Any number possible.	Removes objects from the configuration. It contains the following tags: • <ServiceRefs> • <ActionRefs> • <PropRuleRefs> • <CalcRuleRefs> • <OperatorRefs> • <All>
<SetLabel>	Any number possible.	For internal use only.
<SetAttributes>	Any number possible.	Adds or changes service attributes. It contains the following tags: • <ServiceRef> • <Attribute>

Table 6-3 The Operations Tags (Continued)

Tag	Required?	Description
<RemoveAttributes>	Any number possible.	Removes the specified attributes from the service. It contains the following tags: • <ServiceRef> • <Name>
<RemovePrefAttributes>	Any number possible.	Removes all attributes with the specified name prefix from the service. It contains the following tags: • <ServiceRef> • <Name>
<List>	Any number possible.	Lists objects from the configuration. It contains the following tags: • <Recursive> • <Full> • <Depth> • <ServiceRefs> • <ActionRefs> • <PropRuleRefs> • <CalcRuleRefs> • <OperatorRefs> • <All>
<ListAssignments>	Any number possible.	Lists service-to-operator assignments. It contains the following tags: • <ServiceRef>
<AssignServices>	Any number possible.	Assigns services to an operator. It contains the following tags: • <OperatorRef> • <ServiceRef>
<DeassignServices>	Any number possible.	Deassigns services from an operator. It contains the following tags: • <OperatorRef> • <ServiceRef> • <All>

Table 6-3 The Operations Tags (Continued)

Tag	Required?	Description
<GetElementStatus>	Any number possible.	Gets the status of a service element. It contains the following tags: • <ServiceRef> • <OperatorRef>
<GetAssocStatus>	Any number possible.	Gets the status of an association. It contains the following tags: • <Dependency> • <Composition> • <OperatorAssignment> • <SourceRef> • <TargetRef>
<GetElementMultiStatus>	Any number possible.	Gets the status of a service element for the specified service status calculation. It contains the following tags: • <ServiceRef> • <OperatorRef> • <CalculationId>
<GetAssocMultiStatus>	Any number possible.	Gets the status of an association for the specified service status calculation. It contains the following tags: • <Dependency> • <Composition> • <OperatorAssignment> • <SourceRef> • <TargetRef> • <CalculationId>
<GetRootCauses>	Any number possible.	Gets the root cause for a service problem. It contains the following tags: • <ServiceRef> • <OperatorRef>

Table 6-3 The Operations Tags (Continued)

Tag	Required?	Description
<GetImpacts>	Any number possible.	Gets the services that are impacted by a problem. It contains the following tags: • <ServiceRef> • <NoCause>
<AddLoggings>	Any number possible.	Starts logging of service status. It contains the following tags: • <RegCondition>
<RemoveLoggings>	Any number possible.	Stops logging of service status. It contains the following tags: • <ServiceRef>
<ListLoggings>	Any number possible.	Lists services for which logging is enabled. It contains the following tags: • <ServiceRef>
<Registration>	Any number possible.	Registration. It contains the following tags: • <RegCondition> • <All> • <WithLabelChanges> • <WithAttributeChanges> • <WithMultiStatusChanges>
<RegCondition>	Any number possible.	Registration condition. It contains the following tags: • <ServiceRef> • <OperatorRef> • <Depth> • <Recursive>

Table 6-3 The Operations Tags (Continued)

Tag	Required?	Description
<GetViewsForOriginalId>	Any number possible.	Gets all service hierarchies that contain a service with a specific original ID: • <OriginalIdRef> • <OperatorRef> This only applies to services configured with the HP Operations Service Navigator.
<GetServicesForMessage>	Any number possible.	Gets all services that are mapped to a specific message. This only applies to services configured with the HP Operations Service Navigator.
<Dump>	Any number possible.	Dumps the contents of the service engine. This is useful for troubleshooting purposes.
<GetStatusCalculations>	Any number possible.	Gets the configuration of the service status calculations.
<GetDefaultStatusCalculation>	Any number possible.	Gets the calculation id of the default service status calculation.
<ServicesRef>	Required	Specifies references to services.
<OperatorRef>	Any number possible.	Specifies an operator.
<OriginalIdRef>	Any number possible.	Specifies the original ID of a service. This only applies to services configured with the HP Operations Service Navigator.
<All>	Empty	All allowed objects.
<Full>	Empty	Requests full configuration information including all referenced objects.
<Recursive>	Empty	Recursive registration.
<Depth>>	Empty	Requests configuration information including all referenced objects for the specified number of hierarchical levels in depth.

Table 6-3 The Operations Tags (Continued)

Tag	Required?	Description
<WithLabelChanges>	Empty	For internal use only.
<WithAttributeChanges>	Empty	For internal use only.
<WithMultiStatusChanges>	Any number possible.	Requests status changes for non-default service status calculations.
<NoCause>	Empty	Gets service hierarchy containing only parent services regardless of the problem severity.

See the file `/opt/OV/OpC/examples/services/operations.xml` for an example of an operations XML file.

The Results Tags

The format of the results file is based on the World Wide Web Consortium Extended Markup Language (XML).

The following is the Document Type Definition (DTD) for the results file. Note that the `results.dtd` includes the `loggings.dtd`, the `operations.dtd` and the `service.dtd`. The `service.dtd` is described in more detail in the *Service Navigator Concepts and Configuration Guide*. The `operations.dtd` is described in "The Operations Tags" on page 596. The `loggings.dtd` is described in "The Loggings Tags" on page 617.

```
<!-- XML DTD for service engine results  -->
<!-- Results is the root element         -->

<!ENTITY % LoggingsDTD SYSTEM "loggings.dtd">

%LoggingsDTD;



<!-- Operations into ServiceEngine -->

<!ELEMENT Results (OK
                   | Error
                   | Warning
                   | Assignments
                   | Loggings
                   | StatusChanges
                   | ConfigChanges
                   | LabelChange
                   | AttributeChange
                   | ElementStatus
                   | ElementMultiStatus
                   | AssocStatus
                   | AssocMultiStatus
                   | Services
```

```
                    | MessageServices
                    | OriginalIdViews
                    | StatusCalculations
                    | DefaultStatusCalculation)*>
<!ATTLIST Results xmlns   CDATA #IMPLIED
           version CDATA #IMPLIED>

<!ELEMENT OK            EMPTY>

<!ELEMENT Error         (#PCDATA)>

<!ELEMENT Warning       (#PCDATA)>

<!ELEMENT Assignments   (Assignment*)>
<!ELEMENT StatusChanges (ElementStatusChange
                         | AssocStatusChange
                         | ElementMultiStatusChanges
                         | AssocMultiStatusChanges)*>

<!ELEMENT ConfigChanges EMPTY>

<!ELEMENT Assignment    (ServiceRef, OperatorRef*)>


<!ELEMENT ElementStatusChange ((ServiceRef | OperatorRef),
                               Status, OldStatus)>
<!ELEMENT AssocStatusChange   ((Dependency | Composition
                                | OperatorAssignment),
                               SourceRef, TargetRef,
                               Status, OldStatus)>
```

```
<!ELEMENT ElementMultiStatusChanges ((ServiceRef | OperatorRef),
                                     MultiStatusChange+)>

<!ELEMENT AssocMultiStatusChanges   ((Dependency | Composition |
                                    OperatorAssignment), SourceRef,
                                    TargetRef, MultiStatusChange+)>

<!ELEMENT MultiStatusChange         (CalculationId, Status,
                                    OldStatus)>

<!ELEMENT ElementStatus ((ServiceRef | OperatorRef), Status)>
<!ELEMENT AssocStatus   ((Dependency | Composition
                          | OperatorAssignment),
                         SourceRef, TargetRef,
                         Status)>

<!ELEMENT ElementMultiStatus ((ServiceRef | OperatorRef),
                              MultiStatus+)>

<!ELEMENT AssocMultiStatus   ((Dependency | Composition
                               | OperatorAssignment),
                               SourceRef, TargetRef,
                               MultiStatus+)>

<!ELEMENT LabelChange (ServiceRef, Label)>
<!ELEMENT AttributeChange (ServiceRef, Attribute*,
                           RemovedAttributes?)>
<!ELEMENT RemovedAttributes (Name*)>

<!ELEMENT MessageServices ((ServiceRef, Label)*) >

<!ELEMENT OldStatus     %Severity;>

<!ELEMENT OriginalIdViews (OriginalIdView*) >
```

```
<!ELEMENT OriginalIdView (TopLevelService, Service*) >

<!ELEMENT TopLevelService (Service) >

<!ELEMENT StatusCalculations (CalculationId, Name)* >

<!ELEMENT DefaultStatusCalculation (CalculationId) >


<!-- EOF -->
```

Table 6-4 The Results Tags

Tag	Required?	Description
<Results>	Required.	Is the root element. It contains the following tags: • <OK> • <Error> • <Warning> • <Assignments> • <Loggings> • <StatusChanges> • <ConfigChanges> • <LabelChange> • <AttributeChange> • <ElementStatus> • <ElementMultiStatus> • <AssocStatus> • <AssocMultiStatus> • <Services> • <MessageServices> • <OriginalIdViews> • <StatusCalculations> • <DefaultStatusCalculation>
<OK>	Any number possible.	OK. Indicates successful operation or result.
<Error>	Any number possible.	Error. Indicates that the operation failed. Contains the error text.
<Warning>	Any number possible.	Enhanced Error Handling for the Service Engine.
<Assignments>	Any number possible.	Represents the assignments of a service to operator(s). It contains the following tag: • <Assignment>

Table 6-4 The Results Tags (Continued)

Tag	Required?	Description
<StatusChanges>	Any number possible.	Represents the status changes for a service. It contains the following tags: • <ElementStatusChange> • <AssocStatusChange> • <ElementMultiStatusChanges> • <AssocMultiStatusChanges>
<ConfigChanges>	Any number possible.	Represents any configuration changes. It contains the following tags: • <Service> • <Operator>
<LabelChange>	Any number possible.	For internal use only.
<AttributeChange>	Any number possible.	For internal use only.
<RemovedAttributes>	Optional.	For internal use only.
<Assignment>	Any number possible.	Lists service-to-operator assignments. It contains the following tags: • <ServiceRef> • <OperatorRef>
<ElementStatusChange>	Any number possible.	Gets the status change of a service or operator. It contains the following tags: • <ServiceRef> • <OperatorRef> • <Status> • <OldStatus>
<ElementMultiStatusChanges>	Any number possible.	Gets the status changes of a service or operator for non-default service status calculation modes. It contains the following tags: • <ServiceRef> • <OperatorRef> • <MultiStatusChange>

Table 6-4 The Results Tags (Continued)

Tag	Required?	Description
<AssocStatusChange>	Any number possible.	Gets the status change of a service association. It contains the following tags: • <Dependency> • <Composition> • <OperatorAssignment> • <SourceRef> • <TargetRef> • <Status> • <OldStatus>
<AssocMultiStatusChanges>	Any number possible.	Gets the status changes of a service association for non-default service status calculation modes. It contains the following tags: • <Dependency> • <Composition> • <OperatorAssignment> • <SourceRef> • <TargetRef> • <MultiStatusChange>
<MultiStatusChange>	Any number possible.	Gets the status change of a service or service association for for non-default service status calculation modes. It contains the following tags: • <Status> • <OldStatus> • <CalculationId>
<ElementStatus>	Any number possible.	Gets the status of a service element. It contains the following tags: • <ServiceRef> • <OperatorRef> • <Status>

Table 6-4 The Results Tags (Continued)

Tag	Required?	Description
<ElementMultiStatus>	Any number possible.	Gets the status of a service element for all service status calculation modes. It contains the following tags: • <ServiceRef> • <OperatorRef> • <MultiStatus>
<AssocStatus>	Any number possible.	Gets the status of a service element association. It contains the following tags: • <Dependency> • <Composition> • <OperatorAssignment> • <SourceRef> • <TargetRef> • <Status>
<AssocMultiStatus>	Any number possible.	Gets the status of a service association for all service status calculation modes. It contains the following tags: • <ServiceRef> • <OperatorRef> • <MultiStatus>
<MessageServices>	Any number possible.	Lists all services (with name and label) whose “service name in message” attribute matches the specified “service name” attribute of a message. It contains the following tags: • <ServiceRef> • <Label> This only applies to services configured with the HP Operations Service Navigator.

Table 6-4 The Results Tags (Continued)

Tag	Required?	Description
<OriginalIdViews>	Any number possible.	Lists all service hierarchies that contain a service with the specified original ID. It contains the following tag: • <OriginalIdView> This only applies to services configured with the HP Operations Service Navigator.
<OriginalIdView>	Any number possible.	Service hierarchy that contains a service with the specified original ID. It contains the following tags: • <TopLevelService> • <Service> <TopLevelService> is the topmost service in the service hierarchy. <Service> is the requested service with the specified original ID.
<StatusCalculations>	Required.	Gets the configuration of the service status calculations. It contains the following tag: • <CalculationId> • <Name>
<DefaultStatusCalculation>	Required.	Gets the calculation id of the default service status calculation. It contains the following tag: • <CalculationId>
<TopLevelService>	Required.	Top-level service in a service hierarchy that has a subservice with the specified original ID. It contains the following tag: • <Service>

Table 6-4 The Results Tags (Continued)

Tag	Required?	Description
<OldStatus>	Required.	Displays the previous status of a service or operator. It contains the following tags: • <Normal> • <Warning> • <Minor> • <Major> • <Critical>

See the file /opt/OV/OpC/examples/services/results.xml for an example of a results XML file.

The Loggings Tags

The format of the loggings file is based on the World Wide Web Consortium Extended Markup Language (XML).

The following is the Document Type Definition (DTD) for the loggings file. Note that the `loggings.dtd` includes the `operations.dtd` and the `service.dtd`. The `service.dtd` is described in more detail in the *Service Navigator Concepts and Configuration Guide*, the `operations.dtd` is described in "The Operations Tags" on page 596.

```
<!-- XML DTD for service engine logging repository  -->
<!-- Loggings is the root element                   -->

<!ENTITY % OperationsDTD SYSTEM "operations.dtd">

%OperationsDTD;

<!ELEMENT Loggings      (Logging*)>
<!ELEMENT Logging       (ServiceRef, (Depth | Recursive)?)>
```

Table 6-5 The Loggings Tags

Tag	Required?	Description
`<Loggings>`	Required.	Lists the services for which logging is enabled. It contains the following tags: • `<Logging>`
`<Logging>`	Any number possible.	Lists the services for which logging is enabled. It contains the following tags: • `<ServiceRef>` • `<Depth>` • `<Recursive>`

Return Values

Table 6-6 on page 618 lists the results that can be expected from the operations described in the previous sections. Most operations return OK if the operation was successful; otherwise an error with error text and explanation is returned.

The output of all operations is in the UTF-8 character set.

Table 6-6 Results

Operation	Result
<Add>	OK.
<AddLoggings>	OK.
<AssignServices>	OK.
<GetAssocStatus>	Status of an association.
<GetAssocMultiStatus>	All statuses of an association
<DeassignServices>	OK.
<Dump>	OK.
<GetElementStatus>	Status of an element.
<GetElementMultiStatus>	All statuses of an element.
<GetImpacts>	Services.
<GetRootCauses>	Services.
<GetServicesForMessage>	Services.
<GetViewsForOriginalId>	Service hierarchies with topmost service and its subservice that has the specified original ID.
<GetStatusCalculations>	Service multi status calculation configuration.
<GetDefaultStatusCalculation>	Default multi status calculation configured.
<List>	Services.
<ListAssignments>	Assignments.

Table 6-6 Results (Continued)

Operation	Result
<ListLoggings>	Loggings.
<Remove>	OK.
<RemoveAttributes>	OK.
<RemoveLoggings>	OK.
<RemovePrefAttributes>	OK.
<Replace>	OK.
<SetAttribues>	OK.
<SetLabel>	For internal use only.

Service Engine C++ APIs

The Service Engine APIs are C++ APIs that provide interfaces to access information in the service engine. These interfaces are:

- The Service Operations Interface Classes
- The Registration Interface Classes

To use the APIs, the following is required:

Header file: `/opt/OV/include/opcsvcapi.h`

Shared library: `/opt/OV/lib/libopcsvcapi.sl`

Compiler: ANSI C++ (aCC)

The output of the Service Engine APIs is always in the UTF-8 codeset.

The Classes

The Service Engine APIs comprise the following classes:

`ServiceEngine`	Maintains a connection to the Service Engine. To obtain a `Service` or `Registration` object, an instance of the class `ServiceEngine` is required.
`Service`	Represents a service instance in the Service Engine.
`Severity`	Severity object returned by the service operations interface.
`Registration`	Registration for service status changes. Maintains registration conditions.
`ServiceStatusListener`	Listener for service status changes.
`ServiceStatusChange`	Provides information on service status transitions.

The classes are described in detail in the following sections. See also the man page *opcsvc_api(3)*.

The ServiceEngine Class

Prototype

```
class ServiceEngine
{
 public:
  ServiceEngine();
  ServiceEngine(bool dolnit);
  ServiceEngine(char const *host);
  ServiceEngine(char const *host, int port);
  ServiceEngine(char const *host, int port, bool dolnit);
  Registration *NewRegistration();
  Service      *NewService( char const *svc );
  void Register( Registration *reg );
  void Register();
  void Unregister();
  void Listen( ServiceStatusListener *statusListener = NULL );
  int GetStatusCalcCount();
  int GetStatusCalcId(int index);
  const char * GetStatusCalcName(int index);
  int GetDefaultStatusCalculation();

  void SetAttributes(char const* service, vector<pair<char\
  const*, char const*> > *attributes);
  void RemoveAttributes(char const* service, vector<char\
  const *> * names = 0);
  void RemovePrefAttributes(char const * service, const char\
  * prefix = 0);
};
```

Description

The class ServiceEngine is a high-level class that provides a query interface to get the status of a service as well as other basic attributes and a registration interface for status changes.

It maintains a connection to the Service Engine. To obtain a `Service` or `Registration` object, an instance of the class `ServiceEngine` is required.

Methods

Table 6-7 Methods of the ServiceEngine Class

Method	Description
`NewRegistration()`	Creates and returns a new `Registration` object; must be deleted by the caller.
`GetStatusCalcCount()`	Get number of service status calculation rules.
`GetStatusCalcId()`	Get the nth service status calculation identifier.
`GetStatusCalcName()`	Get the nth service status calculation name.
`GetDefaultStatusCalculation()`	Get the default service status calculation identifier.
`NewService()`	Creates and returns a new `Service` object; must be deleted by the caller.
`Register()`	Register for status change events according to specification by `Registration` object. Subsequent calls of `Register()` replace the previous registration information. To register for *all* status change events, call `Register()` without any parameters.
`Unregister()`	Unregister all objects.
`Listen()`	Listen for status changes. Listen() is a blocking call; it only returns a value if the operation failed. `ServiceStatusChanged()` of `ServiceStatusListener` is called when changes take place.
`SetAttributes()`	Adds or changes service attributes on specified service.
`RemoveAttributes()`	Removes the specified attributes from the specified service.
`RemovePrefAttributes()`	Removes all attributes with the specified name prefix from the specified service.

See the following sections for examples.

The Service Operations Interface Classes

The service operations interface allows you to access a service in the service engine and retrieve its status and/or other properties, as well as to set or remove service attributes. The service operations interface comprises the `Service` and the `Severity` class.

The Service Class

Prototype

```
class Service
{
 public:
  virtual ~Service() {}
  virtual Severity GetStatus() const = 0;
  virtual Severity GetMultiStatus(int calcId) const = 0;
  virtual Severity GetStatus() const = 0;
  virtual char const *GetLabel() const = 0;
  virtual char const *GetDescription() const = 0;
  virtual char const *GetTitle() const = 0;
  virtual char const *GetIcon() const = 0;
  virtual char const *GetBackground() const = 0;
  virtual int GetDepth() const = 0;
  virtual char const *GetMsgPropRuleName() const = 0;
  virtual char const *GetCalcRuleName() const = 0;
  virtual float GetMsgWeight() const = 0;
  virtual char const *GetAttribute( char const *param )\
  const = 0;
  virtual int GetAttrCount() const = 0;
  virtual char const *GetAttrName(int idx) const = 0;
  virtual char const *GetAttrValue(int idx) const = 0;
  virtual int GetNodeCount() const = 0;
  virtual char const *GetNode(int idx) const = 0;
  virtual int GetActionCount() const = 0;
  virtual char const *GetAction(int idx) const = 0;

  virtual void SetAttributes(vector<pair<char const *, char\
  const *> > * attributes) = 0;
  virtual void RemoveAttributes(vector<char const *>\
  * names) = 0;
  virtual void RemovePrefAttributes(char const * prefix=0)= 0;
  virtual int GetMsgSvcNameCount() const = 0;
  virtual char const * GetMsgSvcName(int idx) const = 0;
  virtual char const * GetOriginalId() const = 0;
};
```

Description

The class `Service` represents a service instance in the Service Engine. With `ServiceEngine:NewService()`, a service object can be retrieved and then used to query the status, multi-status and/or other basic properties of the service, as well as to set or remove the service attributes.

NOTE If a `Service` object is not obtained, you can use an instance of the class `ServiceEngine` class to set or remove service attributes instead of using the instance of the class `Service`. For more information about `Service Engine` class, see "The ServiceEngine Class" on page 621.

Methods

Table 6-8 Methods of the Service Class

Method	Description
`GetStatus()`	Get current status of the service. Returns an object of the class `Severity`.
`GetMultiStatus()`	Get current status of the service for the specified service status calculation identifier. Returns an object of the class `Severity`.
`GetLabel()`	Get the label of the service.
`GetDescription()`	Get the description of the service.
`GetTitle()`	Get the title of the service.
`GetIcon()`	Get the icon of the service. This can be either a filename or a URL.
`GetBackground()`	Get the background of the service.
`GetDepth()`	Get the depth value of the service.
`GetMsgPropRuleName()`	Get the name of the propagation rule for messages of the service.
`GetCalcRuleName()`	Get the name of the calculation rule of the service.
`GetMsgSvcNameCount()`	Get number of `Service Name in Message` attributes.

Table 6-8 Methods of the Service Class (Continued)

Method	Description
`GetMsgSvcName()`	Get the nth `Service Name in Message` attribute.
`GetOriginalId()`	Get the identifier set by HP Service Configuration for Service Navigator.
`GetMsgWeight()`	Get the message weight factor of the service.
`GetAttribute()`	Get the name attribute value of the service.
`GetAttrCount()`	Get the number of attributes of the service.
`GetAttrName()`	Get the name of the nth attribute of the service.
`GetAttrValue()`	Get the value of the nth attribute of the service.
`GetNodeCount()`	Get the number of nodes of the service.
`GetNode()`	Get the nth node name of the service.
`GetActionCount()`	Get the number of actions of the service.
`GetAction()`	Get the nth action name of the service.
`SetAttributes()`	Adds or changes service attributes.
`RemoveAttributes()`	Removes the specified attributes from the service.
`RemovePrefAttributes()`	Removes all attributes with the specified name prefix from the service.

The Severity Class

Prototype

```
class Severity
{
 public:
  typedef enum {
   Unused,
   Normal,
   Warning,
   Minor,
   Major,
   Critical,

   COUNT
  } Type;

   Severity();
   Severity( Type sev );
   Severity & operator=( Type sev );
   operator Type() const;
};
```

Description

The `Service::GetStatus()` method of class `Service` returns a `Severity` object that represents the severity of the service. The status value is retrieved each time `Service::GetStatus()` is called whereas the other `Get*()` methods return the values that have been fetched from the engine at object creation time with `ServiceEngine::NewService()`.

Examples

To compile the examples, use the following makefile:

`/opt/OV/OpC/examples/progs/svcapi/Makef.svcapi`

See the following C++ example program:

`/opt/OV/OpC/examples/progs/svcapi/opcsvcget.cpp`

`/opt/OV/OpC/examples/progs/svcapi/opcsvcsev.cpp`

`/opt/OV/OpC/examples/progs/svcapi/opcsvcsevmlt.cpp`

The Registration Interface Classes

This interface allows you to register for service status changes. The information that is returned includes the service name, the previous severity, and the new severity.

The Registration Class

Prototype

```
class Registration
{
 public:
  virtual ~Registration() {}
  virtual void AddRegCond(char const *name, int level = 0) = 0;
  virtual void RemoveRegCond(char const *name = 0) = 0;
  virtual void ReceiveMultiCalculation(bool cond = false) = 0;

};
```

Description

To register for status change events a object of class `Registration` has to be passed to `ServiceEngine::Register()`. A `Registration` maintains registration conditions where each condition specifies a service name and a depth value.

Registration condition also specifies wheather change events are coming also for non-default service status calculation modes. To enable receiving of status changes for non-default service status calculation modes, ReceiveMultiCalculation method with value true must be called on Registration object, before it is passed to ServiceEngine::Register().

Registration conditions can be added and removed. Adding a different registration condition for a service twice replaces the previous registration condition. Each call of `ServiceEngine::Registration()` replaces the previous registration.

Depth values `d` have the following meaning:

d = 0	Recursive. Gets change events for the service and its subservices.
d = 1	Only the service itself. This is the default.
d > 1	d levels below the service itself (including the service); for example, d=2 means the service plus its subservices.

The `ServiceEngine::Listen()` call is a blocking call which only returns a value in case of failure. `Listen()` calls the callback methods `ServiceStatusListener::ServiceStatusChanged()` (and/or `ServiceStatusListener:: ServiceMultiStatusChanged()` if receiving of status changes for non-default status calculation modes was enabled with `ReceiveMultiCalculation` method.

If an API program wants to use both service operations interfaces such as `Service::GetStatus()` and the registration interface, it has to use threads to perform these tasks in parallel.

Table 6-9 Methods of the Registration Class

Method	Description
`AddRegCond()`	Add a registration condition of a service name and depth specification. The default for depth is 1.
`RemoveRegCond()`	Remove a registration condition. Name specifies service name for which a previous registration condition was created.
`ReceiveMultiCalculation()`	Specify weather service status listener receives status changes also for non-default service status calculation modes.

The ServiceStatusListener Class

Prototype

```
class ServiceStatusListener
{
 public:
  virtual ~ServiceStatusListener() {}
  virtual void ServiceStatusChanged
      (vector<ServiceStatusChange *> const & changes) = 0;
  virtual voidServiceMultiStatusChanged
      (vector<ServiceMultiStatusChange *> const & changes) = 0;
};
```

Description

In the `ServiceEngine::Listen()` method, the caller has to pass an object that implements class `ServiceStatusListener`. When a status change occurs for a service the API user has registered for the method `ServiceStatusChanged()` and the interface is called with a vector of `ServiceStatusChange` objects. The same mechanism is used for registering to the Multistatus change events, where the method `ServiceMultistatusChanged()` is invoked with a vector of `ServiceMultistatusChange` objects.

Because this is only an interface the API user has to implement its own class which inherits from `ServiceStatusListener`. This class implements whatever action the API user wants to perform on behalf of a service status change.

The ServiceStatusChange Class

Prototype

```
class ServiceStatusChange
{
 public:
  virtual char const * GetServiceName() const = 0;
  virtual Severity GetOldStatus() const = 0;
  virtual Severity GetStatus() const = 0;
};
```

Description

A `ServiceStatusChange` object contains the information about a status change of one service. Multiple `ServiceStatusChange` objects can be created by one or many changes in the service engine.

Table 6-10 Methods of the Service StatusChange Class

Method	Description
`GetServiceName()`	Get the name of the service whose status has changed.
`GetStatus()`	Get the current status of the service.This is a value of the class `Severity`.
`GetOldStatus()`	Get the old status of the service.This is a value of the class `Severity`. The transition was from the old status to the current status.

The ServiceMultiStatusChange Class

Prototype

```
class ServiceMultiStatusChange
{
 public:
  virtual ~ServiceMultiStatusChange() {}

  // get properties of a status change
  virtual char const * GetServiceName() const = 0;
  virtual int GetCalculationCount() const = 0;
  virtual Severity GetOldStatus(int index) const = 0;
  virtual Severity GetStatus(int index) const = 0;
  virtual int GetCalculationId(int index) const = 0;

};
```

Description

A `ServiceMultiStatusChange` object provides the information on one multistatus change transfer for one service. Multiple `ServiceMultiStatusChange` can be created by one or many changes in the service engine.

Table 6-11 Methods of the ServiceMultiStatusChange Class

Method	Description
`GetServiceName()`	Get the name of the service whose status(es) has changed.
`GetStatus()`	Get the current status of the service for nth service status calculation view. Valid values for index parameter are between 0 and value returned by `GetCalculationCount()` method. This is a value of the class `Severity`.
`GetOldStatus()`	Get the old status of the service for nth service status calculation view. Valid values for index parameter are between 0 and value returned by `GetCalculationCount()` method. This is a value of the class `Severity`. The transition was from the old status to the current status.
`GetCalculationCount()`	Get number of service status calculation modes for which status has changed.

Table 6-11 Methods of the ServiceMultiStatusChange Class (Continued)

Method	Description
`GetCalculationId()`	Get nth service status calculation view identifier. Valid values for index parameter are between 0 and value returned by `GetCalculationCount()` method.

Examples

To compile the examples, use the following makefile:

```
/opt/OV/OpC/examples/progs/svcapi/Makef.svcapi
```

See the following C++ example program:

```
/opt/OV/OpC/examples/progs/svcapi/opcsvcact.cpp
/opt/OV/OpC/examples/progs/svcapi/opcsvclog.cpp
/opt/OV/OpC/examples/progs/svcapi/opcsvclogmlt.cpp
```

Allowing Remote Access to the Service Engine

By default, remote access to the service engine is disabled.

To allow remote access to the service engine, make the following configuration changes:

1. Enter the following line in the `/etc/services` file:

   ```
   opcsvcterm  7278/tcp    # Service engine remote access
   ```

2. Enter the following line in the `/etc/inetd.conf` file:

   ```
   opcsvcterm  stream tcp nowait root
   /opt/OV/bin/OpC/opcsvcterm  opcsvcterm
   ```

3. Restart the `inetd` process:

   ```
   inetd -c
   ```

   ```
   kill -s HUP `ps -e | grep 'inet[d]' | nawk '{print $1}'`
   ```

A About HPOM Man Pages

In this Appendix

This appendix describes the man pages available in the following areas:

- Man Pages in HPOM
- Man Pages for HPOM APIs
- Man Pages for HP Operations Service Navigator

Accessing and Printing Man Pages

You can access the HPOM man pages from the command line, from online help, or in HTML format on your management server.

To Access an HPOM Man Page from the Command Line

To access an HPOM man page from the command line, enter the following:

man <*manpagename*>

To Print a Man Page from the Command Line

To print an HPOM man page from the command line, enter the following:

man <*manpagename*> | col -lb | lp -d *printer_name*

To Access the Man Pages in HTML Format

To access the HPOM man pages in HTML format, from your Internet browser, open the following location:

`http://<management_server>:3443/ITO_MAN`

In this URL, <*management_server*> is the fully qualified hostname of your management server.

Man Pages in HPOM

This section describes man pages in HPOM.

Table A-1 HPOM Man Pages

Man Page	Description
call_sqlplus.sh(1)	Calls SQL*Plus.
inst.sh(1M)	Installs HPOM software on managed nodes.
inst_debug(5)	Debugs an installation of the HP Operations agent software.
ito_op(1M)	Launches the HPOM Java-based operator or Service Navigator GUI.
ito_op_api_cli(1M)	Enables calling the Java GUI Remote APIs.
opcbackup_offline(1M)	Interactively saves the HPOM environment for Oracle.
opcbackup_offline(5)	Backs up the HPOM configuration.
opc_chg_ec(1M)	Changes circuit names in event correlation (EC) policies in the HPOM database.
opcrecover_offline(1M)	Interactively recovers the HPOM environment for Oracle.
opcrecover_offline(5)	Recovers the HPOM configuration.
opcack(1M)	Externally acknowledges active messages.
opcackmsg(1M)	Externally acknowledges active messages using message IDs.
opcackmsgs(1M)	Externally acknowledges active messages using specific message attributes.
opcactivate(1M)	Activates a pre-installed HP Operations agent.
opcadddbf(1M)	Adds a new datafile to an Oracle tablespace.
opcagt(1M)	Administers agent processes on a managed node.

Table A-1 HPOM Man Pages (Continued)

Man Page	Description
opcagtutil(1M)	Parses the agent platform file, and performs operations with extracted data.
opccfgdwn(1M)	Downloads configuration data from the database to flat files.
opccfgout(1M)	Configures condition status variables for scheduled outages in HPOM.
opccfgupld(1M)	Uploads configuration data from flat files into the database.
opccltconfig(1M)	Configures HPOM client filesets.
opcconfig(1M)	Configures an HPOM management server.
opccsa(1M)	Provides the functionality for listing, mapping, granting, denying and deleting specified certificate requests.
opccsacm(1M)	Performs the ovcm's functionality for manually issuing new node certificate and using the installation key.
opcdbidx(1M)	Upgrades the structure of the HPOM database.
opcdbinit(1M)	Initializes the database with the default configuration.
opcdbinst(1M)	Creates or destroys the HPOM database scheme.
opcdbpwd(1M)	Changes the password of the HPOM database user opc_op.
opcdbsetup(1M)	Creates the tables in the HPOM database.
opcdcode(1M)	Views HPOM encrypted policy files.
opcerr(1M)	Displays instruction text for HPOM error messages.
opcgetmsgids(1m)	Gets message IDs to an original message ID.
opchbp(1M)	Switches heartbeat polling of managed nodes on or off.
opchistdwn(1M)	Downloads HPOM history messages to a file.
opchistupl(1M)	Uploads history messages into the HPOM database.

Table A-1 HPOM Man Pages (Continued)

Man Page	Description
opcinstrumcfg(1M)	Manages category info in the filesystem and database level simultaneously.
opcmack(1)	Acknowledges an HPOM message by specifying the message ID.
opcmom(4)	Provides an overview of HPOM MoM functionality.
opcmomchk(1)	Checks syntax of MoM policies.
opcmon(1)	Forwards the value of a monitored object to the HPOM monitoring agent on the local managed node.
opcmsg(1)	Submits a message to HPOM.
opcpat(1)	Tests a program for HPOM pattern matching.
opcragt(1M)	Remotely administers agent services for HPOM on a managed node.
opcskm(3)	Manages secret keys.
opcsqlnetconf(1M)	Configures the HPOM database to use an Net8 connection.
opcsv(1M)	Administers HPOM manager services.
opcsw(1M)	Sets the software status flag in the HPOM database.
opcswitchuser(1M)	Switches the ownership of the HP Operations agents.
opcpolicy(1M)	Maintains policies in files.
opcpolicy(1M)	Enables and disables policies.
opctmpldwn(1M)	Downloads and encrypts HPOM message source policies.
opcwall(1)	Sends a message to currently logged in HPOM users.
ovocomposer(1M)	Performs tasks related to OV Composer.
ovocomposer(5)	Describes the Correlation Composer, an HPOM event correlation feature.

Table A-1 HPOM Man Pages (Continued)

Man Page	Description
`ovtrap2opc(1M)`	Converts the `trapd.conf` file and the HPOM policy file.

Man Pages for HPOM APIs

This section describes man pages for HPOM application program interfaces (APIs).

Table A-2 HPOM API Man Pages

Man Page	Description
`opcmon(3)`	Forwards the value of a monitored object to the HPOM monitoring agent on the local managed node.
`opcmsg(3)`	Submits a message to HPOM.

Man Pages for HP Operations Service Navigator

This section describes man pages for the HP Operations Service Navigator.

Table A-3 Service Navigator Man Pages

Man Page	Description
`opcservice(1M)`	Configures HP Operations Service Navigator.
`opcsvcattr (1M)`	Add, change or remove service attributes.
`opcsvcconv(1M)`	Converts service configuration files of HP Operations Service Navigator from the previous syntax to the Extensible Markup Language (XML).
`opcsvcdwn(1M)`	Downloads service status logs of HP Operations Service Navigator to a file.
`opcsvcterm(1M)`	Emulates an interface to HP Operations Service Navigator. The interface inputs Extensible Markup Language (XML) markup into `stdin` and outputs Extensible Markup Language (XML) markup to `stdout`.
`opcsvcupl(1M)`	Uploads service status logs of HP Operations Service Navigator into the HPOM database.

B API Changes

In this Appendix

This appendix lists the API changes from HPOM 8.xx to 9.00 in the HPOM Developer's Toolkit.

Changes from HPOM 8.xx to 9.00

This section lists the changes in the HPOM Developer's Toolkit between versions 8.xx and 9.00:

- API Changes between 8.xx and 9.00
- Data Structure Changes between 8.xx and 9.00
- APIs Used for Backward Compatibility

API Changes between 8.xx and 9.00

The APIs listed in is Table B-1 have been introduced between HPOM 8.xx and 9.00.

Table B-1 New APIs

API	Version	Description
Server Message API		
`opcmsg_set_owner()`	8.25	The function `opcmsg_set_owner()` sets the owner of an active message to the specified operator. See "opcmsg_set_owner()" on page 152 for more information.
Category Configuration API		
`opcinstrum_add_categories()`	9.00	The function `opcinstrum_add_categories()` is used for adding categories to the HPOM database. See "opcinstrum_add_categories()" on page 227 for more information.
`opcinstrum_del_categories()`	9.00	The function `opcinstrum_del_categories()` is used for removing categories from the HPOM database. See "opcinstrum_del_categories()" on page 229 for more information.

Table B-1 New APIs (Continued)

API	Version	Description
`opcinstrum_get_categories()`	9.00	The function `opcinstrum_get_categories()` is used for getting a list of all categories that exist in the HPOM database. See "opcinstrum_get_categories()" on page 231 for more information.
`opcinstrum_get_category()`	9.00	The function `opcinstrum_get_category()` is used for getting the full configuration of the specified category. See "opcinstrum_get_category()" on page 232 for more information.
`opcinstrum_modify_categories()`	9.00	The function `opcinstrum_modify_categories()` is used for changing name or description of categories in the HPOM database. See "opcinstrum_modify_categories()" on page 233 for more information.
`opcpolicy_assign_categories()`	9.00	The function `opcpolicy_assign_categories()` is used for assigning a list of categories to the specified policy. See "opcpolicy_assign_categories()" on page 235 for more information.
`opcpolicy_deassign_categories()`	9.00	The function `opcpolicy_deassign_categories()` is used for deassigning a list of categories from the specified policy. See "opcpolicy_deassign_categories()" on page 237 for more information.
`opcpolicy_get_categories()`	9.00	The function `opcpolicy_get_categories()` is used for getting a list of all categories that are assigned to the specified policy. See "opcpolicy_get_categories()" on page 239 for more information.
`opcnode_assign_categories()`	9.00	The function `opcnode_assign_categories()` is used for assigning a list of categories to the specified node. See "opcnode_assign_categories()" on page 241 for more information.

Table B-1 New APIs (Continued)

API	Version	Description
`opcnode_deassign_categories()`	9.00	The function `opcnode_deassign_categories()` is used for deassigning a list of categories from the specified node. See “opcnode_deassign_categories()” on page 243 for more information.
`opcnode_get_categories()`	9.00	The function `opcnode_get_categories()` is used for getting a list of categories that are assigned to the specified node. See “opcnode_get_categories()” on page 245 for more information.
Instruction Text Interface API		
`opcinstruction_add()`	8.27	The function opcinstruction_add() adds a new instruction text interface to the HPOM database. See “opcinstruction_add()” on page 249 for more information.
`opcinstruction_del()`	8.27	The function opcinstruction_del() deletes an instruction text interface from the HPOM database. See “opcinstruction_del()” on page 251 for more information.
`opcinstruction_get()`	8.27	The function opcinstruction_get() gets the full configuration of a specified instruction text interface from the HPOM database. See “opcinstruction_get()” on page 252 for more information.
`opcinstruction_modify()`	8.27	The function opcinstruction_modify() modifies the configuration of a specified instruction text interface in the HPOM database. See “opcinstruction_modify()” on page 254 for more information.
Node Configuration API		
`opcnode_modify_defaults()`	8.25	The function opcnode_modify_defaults() changes the default values of the given network and machine type. See “opcnode_modify_defaults()” on page 297 for more information.

Table B-1 New APIs (Continued)

API	Version	Description
Notification Schedule Configuration API		
`opcnotischedule_add()`	8.30	The function opcnotischedule_add() adds a new notification schedule to the HPOM database. See “opcnotischedule_add()” on page 360 for more information.
`opcnotischedule_del()`	8.30	The function opcnotischedule_del() deletes a notification schedule from the HPOM database. See “opcnotischedule_del()” on page 362 for more information.
`opcnotischedule_get()`	8.30	The function opcnotischedule_get() gets the full configuration of a notification schedule from the HPOM database. See “opcnotischedule_get()” on page 364 for more information.
`opcnotischedule_get_list()`	8.30	The function opcnotischedule_get_list() gets a list of notification schedules from the HPOM database. See “opcnotischedule_get_list()” on page 366 for more information.
`opcnotischedule_modify()`	8.30	The function opcnotischedule_modify() modifies the configuration of a notification schedule in the HPOM database. See “opcnotischedule_modify()” on page 367 for more information.
Notification Service Configuration API		
`opcnotiservice_add()`	8.30	The function opcnotiservice_add() adds a new notification service to the HPOM database. See “opcnotiservice_add()” on page 371 for more information.
`opcnotiservice_del()`	8.30	The function opcnotiservice_del() deletes a notification service from the HPOM database. See “opcnotiservice_del()” on page 373 for more information.

Table B-1 New APIs (Continued)

API	Version	Description
`opcnotiservice_get()`	8.30	The function opcnotiservice_get() gets the full configuration of a notification service from the HPOM database. See “opcnotiservice_get()” on page 374 for more information.
`opcnotiservice_get_list()`	8.30	The function opcnotiservice_get_list() gets a list of notification services from the HPOM database. See “opcnotiservice_get_list()” on page 376 for more information.
`opcnotiservice_modify()`	8.30	The function opcnotiservice_modify() modifies the configuration of a notification services in the HPOM database. See “opcnotiservice_modify()” on page 377 for more information.
Trouble Ticket Configuration API		
`opctroubleticket_get()`	8.27	The function opctroubleticket_get() retrieves information about HPOM configuration for trouble ticket systems from the HPOM database. See “opctroubleticket_get()” on page 421 for more information.
`opctroubleticket_set()`	8.27	The function opctroubleticket_set() configures HPOM to forward messages to a trouble ticket system and stores the configuration in the HPOM database. See “opctroubleticket_set()” on page 422 for more information.
Pattern Matching API		
`opcpat_match()`	8.26	The function `opcpat_match()` returns the result of an HPOM pattern-matching operation and lists the matched strings, assigned to variables. See “opcpat_match()” on page 511 for more information.

Data Structure Changes between 8.xx and 9.00

The data structures listed in is Table B-2 have been introduced between HPOM 8.xx and 9.00.

Table B-2 New Data Structures

Data Structure	Version	Description
OPCDTYPE_CATEGORY_INFO	9.00	Category information data structure. See "OPCDTYPE_CATEGORY_INFO" on page 554 for more information.
OPCDTYPE_INSTR_IF	8.27	Instruction text interface data structure. See "OPCDTYPE_INSTR_IF" on page 559 for more information.
OPCDTYPE_NOTI_SCHEDULE	8.30	Notification schedule data structure. See "OPCDTYPE_NOTI_SCHEDULE" on page 582 for more information.
OPCDTYPE_NOTI_SERVICE	8.30	Notification service data structure. See "OPCDTYPE_NOTI_SERVICE" on page 583 for more information.

APIs Used for Backward Compatibility

The following APIs are obsolete in HPOM 9.00 and should be used for backward compatibility only.

opctempl_delete()

```
#include opcsvapi.h

int opctempl_delete (
        const opc_connection   opc_conn,         /* in */
        const opcdata          policy          /* in */
                    );
```

Parameters

opc_conn	Connection to the HPOM database.
policy	Policy specification of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function opctempl_delete() to remove a policy including all conditions from the HPOM database. A policy is specified by either the UUID or the name and type. If the UUID is given, a specified name and type will be ignored.

NOTE Deleting a policy also removes all assignments to managed nodes which results in a changed configuration. You must redistribute the policies to activate your changes, for example using opc_distrib().

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

Versions

OVO A.05.00 and later.

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctempl_get_list()

```
#include opcsvapi.h

int opctempl_get_list (
       const opc_connection   opc_conn,          /* in */
       opcdata                policies          /* out */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
policies	Container of type OPCDTYPE_TEMPLATE_INFO. Policy elements will be appended to already existing elements.

Description

Use the function opctempl_get_list() to get a list of all configured policies and policy groups including name, description, type, and UUID. You need this information to manipulate HPOM policies stored in policy files.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policies is NULL or is of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplfile_add()

```
#include opcsvapi.h

int opctemplfile_add (
        const opc_connection   opc_conn,          /* in */
        const char *           file,              /* in */
        opcdata                policies          /* out */
                        );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`file`	Name of the policy file. The policy file will be checked to ensure that the resulting policy in the database is valid.
`policies`	Container with elements of type OPCDTYPE_TEMPLATE_INFO (containing name, description, and UUID of the added policies).

Description

Use the function `opctemplfile_add()` to add new policies to the HPOM database. The policy names must be unique and must not already exist in the database. Already existing policies are ignored and must be modified using the function opctemplfile_modify().

The function also returns a list of policy IDs in opcdata structures of the type OPCDTYPE_TEMPLATE_INFO. The policy list can then be used to assign the policies to policy groups, node groups, or nodes.

NOTE Each policy in the policy list has the flag `OPCDATA_STATUS` set. If you receive the return value `OPC_ERR_NOT_COMPLETELY_DONE`, check the status flags of the policies to find out which policy was not added.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `file` is NULL or zero length.
`OPC_ERR_INVALID_OUTPARAM`	`policies` is NULL or of incorrect type.

OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_CANT_OPEN_FILE	Path does not exist.
OPC_ERR_NOT_COMPLETELY_DONE	Not all policies could be added; check the status flag of each policy. The status can be one of the following values: • OPC_ERR_INVALID_NAME_LENGTH • OPC_ERR_INVALID_DESCRIPTION_LENGTH • OPC_ERR_INVALID_COMMAND • OPC_ERR_INVALID_INTERVAL • OPC_ERR_INVALID_FILE • OPC_ERR_INVALID_PATH • OPC_ERR_INVALID_MODE • OPC_ERR_INVALID_CHARSET • OPC_ERR_INVALID_EXEC_USER • OPC_ERR_INVALID_PROG_OR_MIB • OPC_ERR_INVALID_MINMAX • OPC_ERR_INVALID_MSG_GENERATION • OPC_ERR_INVALID_TEMPLATE_TYPE
OPC_ERR_SYNTAX_ERROR	Policy description contains syntax errors.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplfile_get()

```
#include opcsvapi.h

int opctemplfile_get (
        const opc_connection   opc_conn,          /* in */
        const opcdata          policy,          /* in */
        const char *           file               /* out */
                        );
```

Parameters

opc_conn	Connection to the HPOM database.
policy	Policy of type OPCDTYPE_TEMPLATE_INFO.
file	Name of the destination policy file.

Description

Use the function opctemplfile_get() to get the details of a policy configuration. A policy is identified the policy name and type. The policy file is specified by its filename.

opctemplfile_get() reads the entire policy configuration including the conditions from the HPOM database, and stores the information in *opccfgupld(1M)* format in a file. You can then modify the configuration by editing this file, and save the modified information in the HPOM database using the function opctemplfile_modify().

Do not use opctemplfile_get() to get the configuration of policy groups.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL policy is NULL or of incorrect type.
OPC_ERR_INVALID_OUTPARAM	file is NULL or zero length.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy is not in the HPOM database.

OPC_ERR_CANT_WRITE_FILE	Cannot create file.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplfile_modify()

```
#include opcsvapi.h

int opctemplfile_modify (
       const opc_connection   opc_conn,          /* in */
       const char *           file,              /* in */
       opcdata                policies          /* out */
                          );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`file`	Name of the policy file. The policy file will be checked to ensure that the resulting policy in the database is valid.
`policies`	List of the modified policies; container of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function `opctemplfile_modify()` to modify already existing policies in the HPOM database. To specify the policy, the name and the policy type in the policy file are used.

The assignments of the policies remain the same; only the parameters and conditions of the policies are changed.

NOTE It is not possible to change the name of the policy because the policy name acts as identifier for the policy.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL, `file` is NULL or zero length, `policies` is NULL or of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.

OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy does not exist in the HPOM database.
OPC_ERR_CANT_OPEN_FILE	Policy files do not exist.
OPC_ERR_NOT_COMPLETELY_DONE	Not all policies could be added; check the status flag of each policy.
OPC_ERR_SYNTAX_ERROR	Policy description contains syntax errors.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplgrp_add()

```
#include opcsvapi.h

int opctemplgrp_add (
          const opc_connection   opc_conn,     /* in */
          opcdata                policy group      /* in/out
*/
                        );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`policy group`	policy group of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function `opctemplgrp_add()` to add a new policy group to the HPOM database. To add a policy group, at least the name `OPCDATA_NAME` must be specified.

Initially, a new policy group does not contain any policies; use the function opctemplgrp_assign_templates() to add policies or policy groups to the group.

opctemplgrp_add() creates a new policy group with a new UUID that is returned in the opcdata `policy group`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `policy group` is NULL or of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_ALREADY_EXISTS`	Policy group already exists in the HPOM database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplgrp_assign_templates()

```
#include opcsvapi.h

int opctemplgrp_assign_templates (
        const opc_connection       opc_conn,    /* in */
        const opcdata              policy group,    /* in */
        const opcdata              policies    /* in/out */
                                  );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`policy group`	Policy group of type OPCDTYPE_TEMPLATE_INFO.
`policies`	Container of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function `opctemplgrp_assign_templates()` to assign policies or policy groups to a given policy group. A policy group is identified either by name or by UUID; if both are given, the UUID supersedes the name.

Note that assigning policies or policy groups to a policy group may change the configuration of nodes that have the policy group assigned.

The status of the assignment will be returned in the `OPCDATA_STATUS` field of each element. Check the status when `opctemplgrp_assign_templates()` returns `OPC_ERR_NOT_COMPLETELY_DONE`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `policy group` is NULL or of incorrect type, `policies` is NULL or of incorrect type,
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_OBJECT_NOT_FOUND	Policy group is not in the HPOM database.
OPC_ERR_NOT_COMPLETELY_DONE	Not all assignments were successful; check the status field of each object in policies, it may contain OPC_ERR_OBJECT_NOT_FOUND.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplgrp_deassign_templates()

```
#include opcsvapi.h

int opctemplgrp_deassign_templates (
       const opc_connection       opc_conn,     /* in */
       const opcdata              policy group,     /* in
*/
       const opcdata              policies     /* in */
              );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`policy group`	Policy group of type OPCDTYPE_TEMPLATE_INFO.
`policies`	Container of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function `opctemplgrp_deassign_templates()` to deassign policies or policy groups from a given policy group. A policy group is identified either by name or by UUID; if both are given, the UUID supersedes the name.

Note that deassigning policies or policy groups from a policy group may change the configuration of nodes that have the policy group assigned. If a policy is not assigned to any other policy group, it is moved to the `Toplevel` policy group.

The status of the deassignment will be returned in the `OPCDATA_STATUS` field of each element. Check the status when `opctemplgrp_assign_templates()` returns `OPC_ERR_NOT_COMPLETELY_DONE`.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `policy group` is NULL or of incorrect type, `policies` is NULL or of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.

OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.
OPC_ERR_OBJECT_NOT_FOUND	Policy group is not in the HPOM database.
OPC_ERR_NOT_COMPLETELY_DONE	Not all assignments were successful; check the status field of each object in policies, it may contain OPC_ERR_OBJECT_NOT_FOUND.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplgrp_delete()

```
#include opcsvapi.h

int opctemplgrp_delete (
        const opc_connection       opc_conn,     /* in */
        const opcdata              policy group      /* in
*/
                );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`policy group`	Policy group of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function `opctemplgrp_delete()` to remove a given policy group from the HPOM database. A policy group is identified either by name or by UUID; if both are given, the UUID supersedes the name.

Note that this function does *not* remove policies or policy groups contained in this group. Policies and policy groups that are not contained in any other policy group are moved to the `Toplevel` policy group.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `policy group` is NULL or of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy group is not in the HPOM database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplgrp_get()

```
#include opcsvapi.h

int opctemplgrp_get (
        const opc_connection   opc_conn,         /* in */
        const opcdata          policy group,         /* in
*/
        opcdata                templ_info        /* out */
              );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`policy group`	Policy group of type OPCDTYPE_TEMPLATE_INFO.
`templ_info`	Policy group of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function `opctemplgrp_get()` to get the configuration of a specified policy group from the HPOM database. A policy group is identified by its name or UUID. The UUID has higher priority.

opctemplgrp_get() returns an opcdata structure of the type OPCDTYPE_TEMPLATE_INFO that can then be used to get the list of assigned policies using the function opctemplgrp_get_templates().

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL, `policy group` is NULL or of incorrect type.
`OPC_ERR_INVALID_OUTPARAM`	`templ_info` is NULL or is of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy group is not in the database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplgrp_get_list()

```
#include opcsvapi.h

int opctemplgrp_get_list (
        const opc_connection       opc_conn,     /* in */
        const opcdata              policy group      /* out
*/
                );
```

Parameters

opc_conn	Connection to the HPOM database.
policy group	Policy group of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function opctemplgrp_get_list() to get a list of all policy groups currently stored in the HPOM database.

The function returns a container with elements of the type OPCDTYPE_TEMPLATE_INFO. The policy group elements are appended to already existing elements.

Return Values:

OPC_ERR_OK	OK
OPC_ERR_INVALID_INPARAM	opc_conn is NULL.
OPC_ERR_INVALID_OUTPARAM	policy group is NULL or of incorrect type.
OPC_ERR_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_ACCESS_DENIED	User is not an administrator.
OPC_ERR_CFGUPLD_RUNNING	API is blocked while opccfgupld is running.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplgrp_get_templates()

```
#include opcsvapi.h

int opctemplgrp_get_templates (
        const opc_connection       opc_conn,     /* in */
        const opcdata              policy group,     /* in
*/
        opcdata                    policies     /* out */
              );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`policy group`	Policy group of type OPCDTYPE_TEMPLATE_INFO.
`policies`	Container of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function `opctemplgrp_get_templates()` to get a list of policies and policy groups assigned to a given policy group. A policy group is identified either by name or by UUID. The UUID supersedes the name if both are given.

The function returns a container with elements of the type OPCDTYPE_TEMPLATE_INFO.

Return Values:

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `policy group` is NULL or of incorrect type.
`OPC_ERR_INVALID_OUTPARAM`	`policies` is NULL or of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.
`OPC_ERR_OBJECT_NOT_FOUND`	Policy group is not in the HPOM database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

opctemplgrp_modify()

```
#include opcsvapi.h

int opctemplgrp_modify (
        const opc_connection       opc_conn,     /* in */
        const opcdata              policy group,     /* in
*/
        const opcdata              mod_templgrp  /* in */
              );
```

Parameters

`opc_conn`	Connection to the HPOM database.
`policy group`	Policy group to be modified; of type OPCDTYPE_TEMPLATE_INFO.
`mod_templgrp`	New policy group definition of type OPCDTYPE_TEMPLATE_INFO.

Description

Use the function `opctemplgrp_modify()` to modify an existing policy group in the HPOM database. To modify a policy group, at least the name `OPCDATA_NAME` must be specified.

You can only modify the name and description; use opctemplgrp_assign_templates() or opctemplgrp_deassign_templates() to change the list of assigned policies and policy groups.

Return Values

`OPC_ERR_OK`	OK
`OPC_ERR_INVALID_INPARAM`	`opc_conn` is NULL `policy group` is NULL or of incorrect type, `mod_templgrp` NULL or of incorrect type.
`OPC_ERR_DATABASE_ERROR`	Access to database failed.
`OPC_ERR_NO_MEMORY`	Allocation of memory failed.
`OPC_ERR_ACCESS_DENIED`	User is not an administrator.
`OPC_ERR_CFGUPLD_RUNNING`	API is blocked while `opccfgupld` is running.

OPC_ERR_OBJECT_NOT_FOUND	Policy group is not in the HPOM database.

Versions

5.00 and later

See Also

“opc_connect()” on page 181

“OPCDTYPE_TEMPLATE_INFO” on page 585

A

accessing
  man pages
    command line, 639
    HTML format, 639
additional documentation, 24
Adobe Portable Document Format. *See* PDF documentation
Agent Message API
  agent configuration, 163
  functions, 162
  opcagtmsg_ack(), 164
  opcagtmsg_send(), 165
  opcmsg(), 167
Agent Monitor API
  functions, 169
  opcagtmon_send(), 170
  opcmon(), 172
APIs
  man pages
    HPOM, 644
Application Configuration API
  functions, 189
  opcappl_add(), 190
  opcappl_delete(), 192
  opcappl_get(), 194
  opcappl_get_list(), 196
  opcappl_modify(), 198
  opcappl_start(), 200
Application Group Configuration API
  functions, 202
  opcapplgrp_add(), 203
  opcapplgrp_assign_applgrps(), 205
  opcapplgrp_assign_appls(), 207
  opcapplgrp_deassign_applgrps(), 209
  opcapplgrp_deassign_appls(), 211
  opcapplgrp_delete(), 213
  opcapplgrp_get(), 215
  opcapplgrp_get_applgrps(), 217
  opcapplgrp_get_appls(), 219
  opcapplgrp_get_list(), 221
  opcapplgrp_modify(), 223

C

C++ classes
  overview, 620
  registration interface class, 628
  Service class, 624
  service operations interface class, 623
  ServiceEngine class, 621
  ServiceMultiStatusChange class, 633
  ServiceStatusChange class, 632
  ServiceStatusListener class, 631
  Severity class, 627, 629
Category Configuration API
  functions, 225
  opcinstrum_add_categories(), 227
  opcinstrum_del_categories(), 229
  opcinstrum_get_categories(), 231
  opcinstrum_get_category(), 232
  opcinstrum_modify_categories(), 233
  opcnode_assign_categories(), 241
  opcnode_deassign_categories(), 243
  opcnode_get_categories(), 245
  opcpolicy_assign_categories(), 235
  opcpolicy_deassign_categories(), 237
  opcpolicy_get_categories(), 239
changes
  A.08.xx to A.09.00, 649
command line
  accessing man pages, 639
compile options, 34
Configuration API, 176
Connection API
  functions, 179
  opc_connect(), 181
  opc_disconnect(), 183
  opcconn_get_capability(), 184
  opcconn_set_capability(), 187
conventions, document, 19

D

Data API
  functions, 41
  HPOM Container, 42
  HPOM Iterator, 43
  opcdata_append_element(), 45
  opcdata_clear(), 46
  opcdata_copy(), 47
  opcdata_copy_info_to_actresp(), 48
  opcdata_create(), 49
  opcdata_delete_element(), 50
  opcdata_free(), 51
  opcdata_generate_id(), 52
  opcdata_get_cma(), 53
  opcdata_get_cmanames(), 55
  opcdata_get_double(), 57
  opcdata_get_element(), 58

opcdata_get_error_msg(), 59
opcdata_get_long(), 60
opcdata_get_str(), 61
opcdata_insert_element(), 62
opcdata_ladd(), 63
opcdata_ldel(), 64
opcdata_lget_len(), 65
opcdata_lget_long(), 66
opcdata_lget_str(), 67
opcdata_lset_long(), 68
opcdata_lset_str(), 69
opcdata_num_elements(), 70
opcdata_remove_cma(), 71
opcdata_report_error(), 72
opcdata_set_cma(), 73
opcdata_set_double(), 74
opcdata_set_long(), 75
opcdata_set_str(), 76
opcdata_type(), 77
registration conditions, 44
Developer's Toolkit documentation, 24
Distribution API, 497
opc_distrib(), 498
document conventions, 19
documentation, related
additional, 24
Developer's Toolkit, 24
ECS Designer, 24
online, 25
PDFs, 21
documentation,related
print, 22
DTDs
loggings.dtd, 592, 617
operations.dtd, 592, 596
results.dtd, 592, 607
services.dtd, 592
XML notation, 592

E

ECS Designer documentation, 24
Event Correlation Service Designer. *See* ECS Designer documentation
example
HPOM Interfaces, 515
modifying HPOM objects, 177
operations.xml, 600
results.xml, 611
Server Message API, 532

H

HP Event Correlation Service Designer. *See* ECS Designer documentation
HP Operations Manager. *See* HPOM
HP partner program, 38
HPOM
man pages, 640
HPOM APIs
internationalization, 37
HPOM Configuration APIs
Application Configuration API, 189
Application Group Configuration API, 202
Category Configuration API, 225
Connection API, 179
Distribution API, 497
Instruction Text Interface Configuration API, 247
Message Group Configuration API, 256
Message Regroup Condition Configuration API, 265
Node Configuration API, 278
Node Hierarchy Configuration API, 321
Notification Schedule Configuration API, 358
Notification Service Configuration API, 369
Pattern Matching API, 510
Policy Configuration API, 379
Trouble Ticket Configuration API, 420
User Configuration API, 458
User Profile Configuration API, 423
HPOM Container, 42
HPOM Data Structures, 541
OPCDTYPE_ACTION_REQUEST, 543
OPCDTYPE_ACTION_RESPONSE, 545
OPCDTYPE_ANNOTATION, 547
OPCDTYPE_APPL_CONFIG, 548
OPCDTYPE_APPL_GROUP, 551
OPCDTYPE_APPLIC, 552
OPCDTYPE_APPLIC_RESPONSE, 553
OPCDTYPE_CATEGORY_INFO, 554
OPCDTYPE_CONFIG_EVENT, 555
OPCDTYPE_CONTAINER, 542
OPCDTYPE_INFORM_USER, 558
OPCDTYPE_INSTR_IF, 559
OPCDTYPE_LAYOUT_GROUP, 560
OPCDTYPE_MESSAGE, 561
OPCDTYPE_MESSAGE_EVENT, 568
OPCDTYPE_MESSAGE_ID, 572
OPCDTYPE_MONITOR_MESSAGE, 573

OPCDTYPE_NODE, 574
OPCDTYPE_NODE_CONFIG, 576
OPCDTYPE_NODE_GROUP, 580
OPCDTYPE_NODEHIER, 581
OPCDTYPE_NOTI_SCHEDULE, 582
OPCDTYPE_NOTI_SERVICE, 583
OPCDTYPE_REGROUP_COND, 584
OPCDTYPE_TEMPLATE_INFO, 585
OPCDTYPE_USER_CONFIG, 586
OPCDTYPE_USER_RESP_ENTRY, 587
opcregcond, 588
HPOM Interfaces
examples, 515
HPOM Iterator, 43
opciter_begin(), 78
opciter_create(), 79
opciter_end(), 80
opciter_free(), 81
opciter_get_pos(), 82
opciter_next(), 83
opciter_nth(), 84
opciter_prev(), 85
opciter_set_pos(), 86
HPOM Operator APIs
Agent Message API, 162
Agent Monitor API, 169
Data API, 41
Interface API, 94
Server Message API, 127
Server Synchronization API, 500
HTML format, accessing man pages, 639

I

include file
on management server, 36
Instruction Text Interface Configuration API
functions, 247
opcinstruction_add(), 249
opcinstruction_del(), 251
opcinstruction_get(), 252
opcinstruction_modify(), 254
Interface API, 94
functions, 94
opcif_close(), 101
opcif_close_and_buffer(), 103
opcif_get_pipe(), 104
opcif_open(), 106
opcif_read(), 112
opcif_register(), 114
opcif_unregister(), 117
opcif_write(), 118
opcreg_copy(), 120
opcreg_create(), 121
opcreg_free(), 122
opcreg_get_long(), 123
opcreg_get_str(), 124
opcreg_set_long(), 125
opcreg_set_str(), 126
registration conditions, 98
security considerations, 99
internationalization
API functions, 37

L

libraries, 34
on management server, 35, 36
link options, 34

M

makefiles, 36
man pages
accessing
command line, 639
HTML format, 639
APIs
HPOM, 644
HPOM, 637–645
printing, 639
Service Navigator, 645
managed nodes
example makefiles, 36
management server
example makefiles, 36
Message Group Configuration API
functions, 256
opcmsggrp_add(), 257
opcmsggrp_delete(), 259
opcmsggrp_get(), 261
opcmsggrp_get_list(), 262
opcmsggrp_modify(), 263
Message Regroup Condition Configuration API
functions, 265
opcmsgregrp_add(), 266
opcmsgregrp_delete(), 268
opcmsgregrp_get(), 270
opcmsgregrp_get_list(), 272
opcmsgregrp_modify(), 274

opcmsgregrp_move(), 276
modify functions, 177

N

Node Configuration API
functions, 278
opcnode_add(), 280
opcnode_assign_templates(), 282
opcnode_deassign_templates(), 284
opcnode_delete(), 286
opcnode_get(), 288
opcnode_get_defaults(), 290
opcnode_get_list(), 292
opcnode_get_templates(), 293
opcnode_modify(), 295
opcnode_modify_defaults(), 297
opcnodegrp_add(), 299
opcnodegrp_assign_nodes(), 301
opcnodegrp_assign_templates(), 303
opcnodegrp_deassign_nodes(), 305
opcnodegrp_deassign_templates(), 307
opcnodegrp_delete(), 309
opcnodegrp_get(), 311
opcnodegrp_get_list(), 313
opcnodegrp_get_nodes(), 315
opcnodegrp_get_templates(), 317
opcnodegrp_modify(), 319
Node Hierarchy Configuration API
functions, 321
opcnodehier_add(), 323
opcnodehier_add_layoutgrp(), 325
opcnodehier_copy(), 327
opcnodehier_delete(), 329
opcnodehier_delete_layoutgrp(), 330
opcnodehier_get(), 332
opcnodehier_get_all_layoutgrps(), 334
opcnodehier_get_all_nodes(), 336
opcnodehier_get_layoutgrp(), 338
opcnodehier_get_layoutgrps(), 340
opcnodehier_get_list(), 342
opcnodehier_get_nodeparent(), 344
opcnodehier_get_nodes(), 346
opcnodehier_modify(), 348
opcnodehier_modify_layoutgrp(), 350
opcnodehier_move_layoutgrp(), 352
opcnodehier_move_layoutgrps(), 354
opcnodehier_move_nodes(), 356
Notification Schedule Configuration API
functions, 358
opcnotischedule_get(), 364
opcnotischedule_get_list(), 366
opcnotischedule_modify(), 367
Notification Schedule Interface Configuration API
opcnotischedule_del(), 362
Notification Service Configuration API
functions, 369
opcnotischedule_add(), 360
opcnotiservice_add(), 371
opcnotiservice_get(), 374
opcnotiservice_get_list(), 376
opcnotiservice_modify(), 377
Notification Service Interface Configuration API
opcnotiservice_del(), 373

O

online documentation
description, 25
opc_connect(), 181
opc_disconnect(), 183
opc_distrib(), 498
opc_inform_user(), 503
opc_version(), 504
opcagtmon_send(), 170
opcagtmsg_ack(), 164
opcagtmsg_send(), 165
opcanno_add(), 129
opcanno_delete(), 131
opcanno_get_list(), 133
opcanno_modify(), 135
opcappl_add(), 190
opcappl_delete(), 192
opcappl_get(), 194
opcappl_get_list(), 196
opcappl_modify(), 198
opcappl_start(), 200
opcapplgrp_add(), 203
opcapplgrp_assign_applgrps(), 205
opcapplgrp_assign_appls(), 207
opcapplgrp_deassign_applgrps(), 209
opcapplgrp_deassign_appls(), 211
opcapplgrp_delete(), 213
opcapplgrp_get(), 215
opcapplgrp_get_applgrps(), 217
opcapplgrp_get_appls(), 219
opcapplgrp_get_list(), 221
opcapplgrp_modify(), 223
opcconn_get_capability(), 184
opcconn_set_capability(), 187
opcdata_append_element(), 45

opcdata_clear(), 46
opcdata_copy(), 47
opcdata_copy_info_to_actresp(), 48
opcdata_create(), 49
opcdata_delete_element(), 50
opcdata_free(), 51
opcdata_generate_id(), 52
opcdata_get_cma(), 53
opcdata_get_cmanames(), 55
opcdata_get_double(), 57
opcdata_get_element(), 58
opcdata_get_error_msg(), 59
opcdata_get_long(), 60
opcdata_get_str(), 61
opcdata_insert_element(), 62
opcdata_ladd(), 63
opcdata_ldel(), 64
opcdata_lget_len(), 65
opcdata_lget_long(), 66
opcdata_lget_str(), 67
opcdata_lset_long(), 68
opcdata_lset_str(), 69
opcdata_num_elements(), 70
opcdata_remove_cma(), 71
opcdata_report_error(), 72
opcdata_set_cma(), 73
opcdata_set_double(), 74
opcdata_set_long(), 75
opcdata_set_str(), 76
opcdata_type(), 77
OPCDTYPE_ACTION_REQUEST, 543
OPCDTYPE_ACTION_RESPONSE, 545
OPCDTYPE_ANNOTATION, 547
OPCDTYPE_APPL_CONFIG, 548
OPCDTYPE_APPL_GROUP, 551
OPCDTYPE_APPLIC, 552
OPCDTYPE_APPLIC_RESPONSE, 553
OPCDTYPE_CATEGORY_INFO, 554
OPCDTYPE_CONFIG_EVENT, 555
OPCDTYPE_CONTAINER, 542
OPCDTYPE_INFORM_USER, 558
OPCDTYPE_INSTR_IF, 559
OPCDTYPE_LAYOUT_GROUP, 560
OPCDTYPE_MESSAGE, 561
OPCDTYPE_MESSAGE_EVENT, 568
OPCDTYPE_MESSAGE_GROUP, 571
OPCDTYPE_MESSAGE_ID, 572
OPCDTYPE_MONITOR_MESSAGE, 573
OPCDTYPE_NODE, 574
OPCDTYPE_NODE_CONFIG, 576
OPCDTYPE_NODE_GROUP, 580
OPCDTYPE_NODEHIER, 581
OPCDTYPE_NOTI_SCHEDULE, 582
OPCDTYPE_NOTI_SERVICE, 583
OPCDTYPE_REGROUP_COND, 584
OPCDTYPE_TEMPLATE_INFO, 585
OPCDTYPE_USER_CONFIG, 586
OPCDTYPE_USER_RESP_ENTRY, 587
opcif_close(), 101
opcif_close_and_buffer(), 103
opcif_get_pipe(), 104
opcif_open()
  function, 106
  parameters, 106
opcif_read(), 112
opcif_register()
  function, 114
  parameters, 114
opcif_unregister(), 117
opcif_write(), 118
opcinstruction_add(), 249
opcinstruction_del(), 251
opcinstruction_get(), 252
opcinstruction_modify(), 254
opcinstrum_add_categories(), 227
opcinstrum_del_categories(), 229
opcinstrum_get_categories(), 231
opcinstrum_get_category(), 232
opcinstrum_modify_categories(), 233
opciter_begin(), 78
opciter_create(), 79
opciter_end(), 80
opciter_free(), 81
opciter_get_pos(), 82
opciter_next(), 83
opciter_nth(), 84
opciter_prev(), 85
opciter_set_pos(), 86
opcmon(), 172
opcmsg(), 167
opcmsg_ack(), 137
opcmsg_disown(), 139
opcmsg_get(), 141
opcmsg_get_instructions(), 143
opcmsg_get_msgids(), 144
opcmsg_modify(), 146
opcmsg_own(), 148
opcmsg_select(), 150
opcmsg_set_owner(), 152
opcmsg_start_auto_action(), 154
opcmsg_start_op_action(), 156
opcmsg_unack(), 158
opcmsg_unbuffer(), 160
opcmsggrp_add(), 257
opcmsggrp_delete(), 259
opcmsggrp_get(), 261
opcmsggrp_get_list(), 262

opcmsggrp_modify(), 263
opcmsgregrp_add(), 266
opcmsgregrp_delete(), 268
opcmsgregrp_get(), 270
opcmsgregrp_get_list(), 272
opcmsgregrp_modify(), 274
opcmsgregrp_move(), 276
opcnode_add(), 280
opcnode_assign_categories(), 241
opcnode_assign_templates(), 282
opcnode_deassign_categories(), 243
opcnode_deassign_templates(), 284
opcnode_delete(), 286
opcnode_get(), 288
opcnode_get_categories(), 245
opcnode_get_defaults(), 290
opcnode_get_list(), 292
opcnode_get_templates(), 293
opcnode_modify(), 295
opcnode_modify_defaults(), 297
opcnodegrp_add(), 299
opcnodegrp_assign_nodes(), 301
opcnodegrp_assign_templates(), 303
opcnodegrp_deassign_nodes(), 305
opcnodegrp_deassign_templates(), 307
opcnodegrp_delete(), 309
opcnodegrp_get(), 311
opcnodegrp_get_list(), 313
opcnodegrp_get_nodes(), 315
opcnodegrp_get_templates(), 317
opcnodegrp_modify(), 319
opcnodehier_add(), 323
opcnodehier_add_layoutgrp(), 325
opcnodehier_copy(), 327
opcnodehier_delete(), 329
opcnodehier_delete_layoutgrp(), 330
opcnodehier_get(), 332
opcnodehier_get_all_layoutgrps(), 334
opcnodehier_get_all_nodes(), 336
opcnodehier_get_layoutgrp(), 338
opcnodehier_get_layoutgrps(), 340
opcnodehier_get_list(), 342
opcnodehier_get_nodeparent(), 344
opcnodehier_get_nodes(), 346
opcnodehier_modify(), 348
opcnodehier_modify_layoutgrp(), 350
opcnodehier_move_layoutgrp(), 352
opcnodehier_move_layoutgrps(), 354
opcnodehier_move_nodes(), 356
opcnotischedule_add(), 360
opcnotischedule_del(), 362
opcnotischedule_get(), 364
opcnotischedule_get_list(), 366
opcnotischedule_modify(), 367
opcnotiservice_add(), 371
opcnotiservice_del(), 373
opcnotiservice_get(), 374
opcnotiservice_get_list(), 376
opcnotiservice_modify(), 377
opcpat_match(), 511
opcpolicy_assign_categories(), 235
opcpolicy_deassign_categories(), 237
opcpolicy_get_categories(), 239
opcprofile_add(), 424
opcprofile_assign_applgrps(), 426
opcprofile_assign_appls(), 428
opcprofile_assign_profiles(), 430
opcprofile_assign_resps(), 432
opcprofile_deassign_applgrps(), 434
opcprofile_deassign_appls(), 436
opcprofile_deassign_profiles(), 438
opcprofile_deassign_resps(), 440
opcprofile_delete(), 442
opcprofile_get(), 444
opcprofile_get_applgrps(), 446
opcprofile_get_appls(), 448
opcprofile_get_list(), 450
opcprofile_get_profiles(), 452
opcprofile_get_resps(), 454
opcprofile_modify(), 456
opcreg_copy(), 87, 120
opcreg_create(), 88, 121
opcreg_free(), 89, 122
opcreg_get_long(), 90, 123
opcreg_get_str(), 91, 124
opcreg_set_long(), 92, 93, 125
opcreg_set_str(), 126
opcregcond, 588
opcsvcterm, 591
opcsync_inform_server(), 505
opctempl_delete(), 381, 382, 383, 385, 386, 388, 390, 392, 394, 395, 397, 399, 400, 402, 403, 405, 407, 409, 410, 412, 414, 416, 418, 655
opctempl_get_list(), 657
opctemplfile_add(), 658
opctemplfile_get(), 660
opctemplfile_modify(), 662
opctemplgrp_add(), 664
opctemplgrp_assign_templates(), 666
opctemplgrp_deassign_templates(), 668
opctemplgrp_delete(), 670
opctemplgrp_get(), 672
opctemplgrp_get_list(), 674
opctemplgrp_get_templates(), 675
opctemplgrp_modify(), 677
opctransaction_commit(), 507
opctransaction_rollback(), 508

opctransaction_start(), 509
opctroubleticket_get(), 421
opctroubleticket_set(), 422
opcuser_add(), 459
opcuser_assign_applgrps(), 461
opcuser_assign_appls(), 463
opcuser_assign_nodehier(), 465
opcuser_assign_profiles(), 467
opcuser_assign_resps(), 469
opcuser_deassign_applgrps(), 471
opcuser_deassign_appls(), 473
opcuser_deassign_profiles(), 475
opcuser_deassign_resps(), 477
opcuser_delete(), 479
opcuser_get(), 481
opcuser_get_applgrps(), 483
opcuser_get_appls(), 485
opcuser_get_list(), 487
opcuser_get_nodehier(), 489
opcuser_get_profiles(), 491
opcuser_get_resps(), 493
opcuser_modify(), 495

P

partner program, 38
  HP OpenView premier partner, 38
Pattern Matching API, 510
  opcpat_match(), 511
PDF documentation, 21
Portable Document Format. *See* PDF documentation
premier partner, 38
print documentation, 22
printing
  man pages, 639

R

registration condition, 588
registration conditions, 44, 98
  opcreg_copy(), 87
  opcreg_create(), 88
  opcreg_free(), 89
  opcreg_get_long(), 90
  opcreg_get_str(), 91
  opcreg_set_long(), 92, 93
related documentation
  additional, 24
  Developer's Toolkit, 24
  ECS Designer, 24
  online, 25
  PDFs, 21
  print, 22
return values
  XML data interface, 618
rules
  transaction, 502

S

security considerations, 99
Server Message API
  example, 532
  functions, 127
  opcanno_add(), 129
  opcanno_delete(), 131
  opcanno_get_list(), 133
  opcanno_modify(), 135
  opcmsg_ack(), 137
  opcmsg_disown(), 139
  opcmsg_get(), 141
  opcmsg_get_instructions(), 143
  opcmsg_get_msgis(), 144
  opcmsg_modify(), 146
  opcmsg_own(), 148
  opcmsg_select(), 150
  opcmsg_set_owner(), 152
  opcmsg_start_auto_action(), 154
  opcmsg_start_op_action(), 156
  opcmsg_unack(), 158
  opcmsg_unbuffer(), 160
Server Synchronization API
  functions, 500
  opc_inform_user(), 503
  opc_version(), 504
  opcsync_inform_server(), 505
  opctransaction_commit(), 507
  opctransaction_rollback(), 508
  opctransaction_start(), 509
  transaction concept, 501
  transaction rules, 502
Service C++ class, 624
Service Engine APIs
  C++ APIs, 620
Service Navigator
  interfaces and APIs, 590
  Service Engine APIs, 620
  XML data interface, 591
Service Navigator man pages, 645
ServiceEngine APIs
  C++ classes, 620
  compiler, 620

header file, 620
shared library, 620
ServiceEngine C++ class, 621
ServiceMultiStatusChange C++ class, 633
ServiceStatusChange C++ class, 632
ServiceStatusListener C++ class, 631
setlocale(), 37
Severity C++ class, 627, 629
socket for XML data interface, 591

T

Template Configuration API
functions, 379
opctempl_delete(), 381, 382, 383, 385, 386, 388, 390, 392, 394, 395, 397, 399, 400, 402, 403, 405, 407, 409, 410, 412, 414, 416, 418, 655
opctempl_get_list(), 657
opctemplfile_add(), 658
opctemplfile_get(), 660
opctemplfile_modify(), 662
opctemplgrp_add(), 664
opctemplgrp_assign_templates(), 666
opctemplgrp_deassign_templates(), 668
opctemplgrp_delete(), 670
opctemplgrp_get(), 672
opctemplgrp_get_list(), 674
opctemplgrp_get_templates(), 675
opctemplgrp_modify(), 677
transaction concept, 501
transaction rules, 502
Trouble Ticket Configuration API
functions, 420
opctroubleticket_get(), 421
opctroubleticket_set(), 422
typographical conventions. *See* document conventions

U

User Configuration API
functions, 458
opcuser_add(), 459
opcuser_assign_applgrps(), 461
opcuser_assign_appls(), 463
opcuser_assign_nodehier(), 465
opcuser_assign_profiles(), 467
opcuser_assign_resps(), 469
opcuser_deassign_applgrps(), 471
opcuser_deassign_appls(), 473
opcuser_deassign_profiles(), 475
opcuser_deassign_resps(), 477
opcuser_delete(), 479
opcuser_get(), 481
opcuser_get_applgrps(), 483
opcuser_get_appls(), 485
opcuser_get_list(), 487
opcuser_get_nodehier(), 489
opcuser_get_profiles(), 491
opcuser_get_resps(), 493
opcuser_modify(), 495
User Profile Configuration API
functions, 423
opcprofile_add(), 424
opcprofile_assign_applgrps(), 426
opcprofile_assign_appls(), 428
opcprofile_assign_profiles(), 430
opcprofile_assign_resps(), 432
opcprofile_deassign_applgrps(), 434
opcprofile_deassign_appls(), 436
opcprofile_deassign_profiles(), 438
opcprofile_deassign_resps(), 440
opcprofile_delete(), 442
opcprofile_get(), 444
opcprofile_get_applgrps(), 446
opcprofile_get_appls(), 448
opcprofile_get_list(), 450
opcprofile_get_profiles(), 452
opcprofile_get_resps(), 454
opcprofile_modify(), 456

X

XML data interface
filesystem socket, 591
loggings tags, 617
loggings.dtd, 617
opcsvcterm, 591
operations example, 600
operations tags, 596, 599
operations.dtd, 596
results example, 611
results tags, 607, 611
results.dtd, 607
return values, 618
Service Navigator, 591
XML notation, 592
XML notation, 592