HP OpenView Operations Developer's Toolkit

Developer's Reference

Software Version: A.08.10

UNIX

Manufacturing Part Number: B7492-90010

September 2004

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 in 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.

Trademark Notices.

Adobe® is a trademark of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Intel386, Intel80386, Intel486 , and Intel80486 are U.S. trademarks of Intel Corporation.

Contents

Contents

3. Functions of the OVO Configuration APIs

Contents

Contents

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: February 1999

Second Edition: September 1999

Third Edition: June 2000

Fourth Edition: January 2002

Fifth Edition: May 2004

Sixth Edition: September 2004

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 *OVO 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].

OVO Documentation Map

HP OpenView Operations (OVO) provides a set of manuals and online help that help you use the product and 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 manuals are available as Adobe Portable Document Format (PDF) files in the documentation directory on the OVO product CD-ROM.

With the exception of the *OVO Software Release Notes*, all manuals are also available in the following OVO 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 English and `japanese` for Japanese environments.

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

`http://ovweb.external.hp.com/lpe/doc_serv`

Watch this website regularly for the latest edition of the OVO 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.

OVO Manuals

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

Table 2 OVO Manuals

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

Table 2 OVO Manuals (Continued)

Manual	Description	Media
OVO Java GUI Operator's Guide	Provides you with a detailed description of the OVO Java-based operator GUI and Service Navigator. This manual contains detailed information about general OVO and Service Navigator concepts and tasks for OVO 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 OpenView Service Navigator. This manual also contains a high-level overview of the concepts behind service management.	Hardcopy PDF
OVO 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
OVO Supplementary Guide to MPE/iX Templates	Describes the message source templates that are available for MPE/iX managed nodes. This guide is not available for OVO on Solaris.	PDF only
Managing Your Network with HP OpenView Network Node Manager	Designed for administrators and operators. This manual describes the basic functionality of HP OpenView Network Node Manager, which is an embedded part of OVO.	Hardcopy PDF
OVO Database Tuning	This ASCII file is located on OVO management server on the following location: `/opt/OV/ReleaseNotes/opc_db.tuning`	ASCII

Additional OVO-related Products

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

Table 3 Additional OVO-related Manuals

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

OVO Online Information

The following information is available online.

Table 4 OVO Online Information

Online Information	Description
HP OpenView Operations Administrator's Guide to Online Information	Context-sensitive help system contains detailed help for each window of the OVO administrator Motif GUI, as well as step-by-step instructions for performing administrative tasks.
HP OpenView Operations Operator's Guide to Online Information	Context-sensitive help system contains detailed help for each window of the OVO operator Motif GUI, as well as step-by-step instructions for operator tasks.
HP OpenView Operations Java GUI Online Information	HTML-based help system for the OVO Java-based operator GUI and Service Navigator. This help system contains detailed information about general OVO and Service Navigator concepts and tasks for OVO operators, as well as reference and troubleshooting information.
HP OpenView Operations Man Pages	Manual pages available online for OVO. 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 OVO HTTPS-agent are installed on each managed node.

1 Introduction

In this Chapter

This chapter provides information about:

- The OVO Application Programming Interfaces
- Function-naming Conventions
- Libraries for OVO Integrations
- Using APIs in Internationalized Environments
- HP OpenView Partnerships

The OVO Application Programming Interfaces

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

- **OVO Operator API**

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

  The functions for using the **OVO Interfaces** are also part of this API, because the interfaces are also used to get and modify OVO events. See "Interface API" on page 97 for more information.

  See also Chapter 2, "Functions of the OVO Operator APIs," on page 43 for detailed information about each function.

- **OVO Configuration API**

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

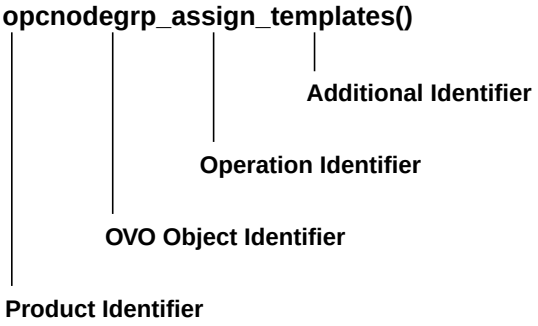
  See also Chapter 3, "Functions of the OVO Configuration APIs," on page 163 for detailed information about each function.

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

Function-naming Conventions

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

Figure 1-1 Naming the OVO API Functions



The function names consist of the following parts:

product identifier	Identifies the product, in OVO this is always opc.
OVO object identifier	Identifies the OVO 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 on page 31 gives an overview of all available identifiers.

NOTE Not all operations are available on all OVO objects and not every addition is available for each operation.

Table 1-1 Overview of Function-name Combinations

Product Identifier	OVO Object Identifier	Operation Identifier	Additional Identifier	Additional Identifier
opc	appl	_add	_all	_layoutgrps
	applgrp	_modify	_list	_nodes
	data	_delete	_node	
	if	_get	_nodes	
	msg	_assign	_nodegrps	
	msggrp	_deassign	_templates	
	msgregrp	_move	_templgrps	
	node		_nodehier	
	nodegrp		_layoutgrp	
	nodehier		_layoutgrps	
	profile		_appls	
	reg		_applgrps	
	sync		_parentusers	
	templ		_profiles	
	templfile		_resps	
	templgrp		_defaults	
	transact			
	user			

Table 1-2, “OVO Objects,” on page 32 gives an overview of all available OVO objects which can be manipulated with the APIs. The opcdata type must be used to describe the objects when using the APIs. See “OVO Data Structures” on page 445 for more information about the opcdata types.

Table 1-2 OVO **Objects**

OVO 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 OVO	OPCDTYPE_APPLIC
Application Configuration	Configuration of an OVO application. This object type is used to configure OVO applications.	OPCDTYPE_APPL_CONFIG
Application Group	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 OVO application. Application responses can be received using the Application Response Interface; see also “Interface API” on page 97.	OPCDTYPE_APPLIC_RESPONSE
Container	A container contains a list of objects of one type.	OPCDTYPE_CONTAINER
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

Table 1-2 OVO **Objects (Continued)**

OVO Object	Description	opcdata Type
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 OVO managed node.	OPCDTYPE_NODE
Node Configuration	A node configuration is the configuration of an OVO 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
Regroup Condition	A regroup condition regroups messages matching the specified condition.	OPCDTYPE_REGROUP_COND
Template	A template is used to configure message conditions on managed nodes.	OPCDTYPE_TEMPLATE_INFO
Template Group	A template group collects several templates and other template groups. Template groups are handled like templates.	OPCDTYPE_TEMPLATE_INFO

Table 1-2 OVO **Objects (Continued)**

OVO Object	Description	opcdata Type
Template File	A template file contains the complete configuration of a template including its conditions. Template files are only used by the template file API.	[char *]
Template Info	A template info object contains the name, description, and type of a template. It can be used to get a list of all available templates instead of the complete template configuration.	`OPCDTYPE_TEMPLATE_INFO`
User Configuration	A user configuration contains the properties of an OVO 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 OVO Integrations

NOTE

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

See the *OVO DCE 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 1-3 Libraries for the Management Server on HP-UX 11.x

OVO Version	OVO A.05.xx	OVO A.06.xx	OVO A.07.xx
Library	**libopcsv_r.sl**	**libopcsv_r.sl**	**libopcsv_r.sl**
Libraries linked to the OVO library.	/usr/lib/libdcekt.1 /usr/lib/libc.2 libnspsv.sl libopcora.sl libpthread.1 libnsl.1 libm.2 (sysliblist) libcl.2 (sysliblist) librt.2 (sysliblist)	/usr/lib/libdcekt.1 /usr/lib/libc.2 libnspsv.sl libpthread.1 libnsl.1 libm.2 (sysliblist) libcl.2 (sysliblist) librt.2 (sysliblist)	/usr/lib/libdcekt.1 libpthread.1 /usr/lib/libnsl.1
Link and compile options	-lopcsv_r -lopcdb	-lopcsv_r -lopcdb -lnspsv	-lopcsv_r -lopcdb -lnspsv
Description	**Note**: Because of the new object format, the new compiler, and the kernel threads on HP-UX 11.x it is not possible to reuse applications linked with OVO A.05.xx on HP-UX 10.x or OVO A.04.xx on HP-UX 10.x/11.x without re-compile and re-link.	n/a	n/a

Table 1-4 Libraries for the OVO Management Server on Sun Solaris

OVO Version	OVO A.05.30	OVO A.06.xx	OVO A.07.xx
OVO libraries	**libopcsv_r.so** **libopcdb.so** **libnspsv.so**	**libopcsv_r.so** **libopcdb.so** **libnspsv.so**	**libopcsv_r.so** **libopcdb.so** **libnspsv.so**
Libraries linked to the OVO libraries	libdce.so libdcecrypt.so libnsl.so.1 libsocket.so.1 libdl.so.1 libc.so.1 libmp.so.1 libclntsh.so.1.0	libdce.so libdcecrypt.so libnsl.so.1 libsocket.so.1 libdl.so.1 libc.so.1 libaio.so.1 libm.so.1 libmp.so.1 libclntsh.so.1.0	libdce.so libdcecrypt.so libnsl.so.1 libsocket.so.1 libdl.so.1 libc.so.1 libaio.so.1 libm.so.1 libmp.so.1 libclntsh.so.1.0 libclntsh.so.8.0 libopcassv.so libgen.so.1 libsched.so.1 libwtc8.so libaio.so.1 libthread.so.1

Table 1-4 Libraries for the OVO Management Server on Sun Solaris

OVO Version	OVO A.05.30	OVO A.06.xx	OVO A.07.xx

Link and compile options	`-mt`
	`-R/opt/OV/lib`
	`-L/opt/OV/lib`
	`-lopcsv_r`
	`-lopcdb`
	`-Bstatic -lCrun`
	`-Bdynamic`
	`-lnspsv`
	`-lclntsh`
		`-mt`
		`-R/opt/OV/lib`
		`-L/opt/OV/lib`
		`-lopcsv_r`
		`-lopcdb`
		`-lnspsv`
		`-lCrun`
			`-mt`
			`-R/opt/OV/lib`
			`-L/opt/OV/lib`
			`-lopcsv_r`
			`-lopcdb`
			`-lnspsv`
			`-lCrun`

Description	Supported compiler: Sun Workshop Compiler C/C++ 5.0. It is possible to use other ANSI compliant C/C++ compilers with appropriate makefile modification, but this is not officially supported. libCrun library must be linked statically because it does not exist in Solaris 2.6 distribution; it is part of Sun Workshop installation. Oracle 8.0.6 libclntsh.so should be used.	Supported compiler: Sun Workshop Compiler C/C++ 5.0. It is possible to use other ANSI compliant C/C++ compilers with appropriate makefile modification, but this is not officially supported. libCrun.so.1 is distributed as part of Solaris 7 and Solaris 8 OS, therefore it is possible to link it dynamically.	Supported compiler: Sun Workshop Compiler C/C++ 5.0. It is possible to use other ANSI compliant C/C++ compilers with appropriate makefile modification, but this is not officially supported. libCrun.so.1 is distributed as part of Solaris 7 and Solaris 8 OS, therefore it is possible to link it dynamically.

Libraries on the Managed Nodes

NOTE Integrations with OVO A.06.xx managed nodes will also work with OVO A.08.10 agent software, if they have been linked using the link options for OVO A.06.xx given in the previous tables.

Instrumentation programs which use the OVO agent APIs must be developed on a system with an OVO agent installed, so that the OVO shared library and `opcapi.h` header files are both available.

On platforms with both an NCS and a DCE agent, the integrator must create two versions of the instrumentation binary: one that runs with the OVO NCS agent and one that runs with the OVO DCE agent. This means that the integrator must install both versions of the OVO agent on his development system. Note, however, that the two versions of OVO agent cannot run simultaneously on the same system.

On these platforms, it is *not* possible to build only one instrumentation binary that runs on both versions of the agent. For example, it is not supported to build an NCS binary and run it on the DCE agent. Binaries created with the NCS OVO library only work on an NCS OVO managed node; binaries created with the DCE OVO library only work on a DCE OVO managed node.

Platforms that support multi-threaded environments, for example DCE, must also supply reentrant system calls that work in this environment. Some platforms only supply reentrant libraries which also work for single-threaded applications. Some have separate libraries—a standard library and a reentrant library; for example, `libc` and `libc_r`, or `libsocket` and `libsocket_r`.

On platforms with two sets of libraries, it is important to link the application using the standard library to the `crt0` object file, and the reentrant library using the `crt0_r` object file. `crt0` and `crt0_r` contain code that is executed before `main()` and is responsible for setting up or initializing the environment before calling any of the library APIs. Mixing reentrant and non-reentrant `crt0` and libraries is not allowed.

The OVO agent for DCE is a multi-threaded application and thus requires and uses reentrant libraries. Consequently, the OVO library `libopc_r` is using reentrant calls from various libraries. To use the APIs in the OVO library, they must be linked correctly. Applications that use

the DCE OVO library must be linked as a multi-threaded application; see the *OVO DCE Agent Concepts and Configuration Guide* for details for each managed node platform. It is not allowed to use non-reentrant applications linked with the NCS OVO library with a DCE OVO agent.

Include Files

NOTE See "Libraries for OVO Integrations" on page 35 for important information about platforms that support both the NCS and the DCE OVO agent.

The OVO include file on the management server:

`/opt/OV/include/opcsvapi.h`

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

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

Makefiles

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

Management Server Makefile

- `Makef.hpsv` (makefile for the management server on HP-UX)
- `Makef.solarissv` (makefile for the management server on Sun Solaris)

Managed Nodes Makefiles

See the *OVO DCE Agent Concepts and Configuration Guide* for the location of the makefiles on all managed node platforms.

Using APIs in Internationalized Environments

All OVO 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.

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

HP OpenView Partnerships

The major benefit resulting from an integration with OVO is the increased customer value of the integrated solution. OVO 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 OVO, 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 OpenView Premier Partner**. Validation ensures that the integration is well-behaved and does not conflict with other integrated solutions. As an HP OpenView Premier Partner, your solution is recommended by HP sales channels, you can leverage from the well-established HP OpenView brand name, and you receive immediate market exposure for your solution through HP market awareness and selling tools.

For more information about the HP OpenView partner programs, see our web site at `http://openview.hp.com`, and select `partners`.

HP OpenView Developer Assist

HP OpenView Developer Assist support that increases the speed, ease, and cost effectiveness of integrating with OVO. For additional documentation and ordering information, see our web site at `http://www.openview.hp.com`, select `partners`, `developers' and third-party applications`, and `developer support services`.

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

- ❏ OVO Data Structures

  (See “OVO Data Structures” on page 445 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 “OVO Data Structures” on page 445.

Multithread Usage

All functions of the OVO Data API are safe to be called by multithreaded applications, and are thread-safe for both POSIX Threads and DCE User Threads. They are neither async-cancel, async-signal, nor fork-safe, and cannot be safely called kernel threads.

The OVO 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 OVO 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 OVO 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.

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.

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 OVO Container” on page 46.

opcdata_append_element()

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

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

Parameters

container	OVO data structure of type OPCDTYPE_CONTAINER.
element	OVO 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

OVO A.04.00 and later

See Also

“OVO Data Structures” on page 445

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 an 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

OVO A.05.00 and later

opcdata_copy()

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

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

Parameters

data	OVO 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

OVO A.02.00 and later

See Also

“OVO Data Structures” on page 445

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	OVO 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 data->type. If data points to a message, it is assumed that a response for a local automatic action is to produce, if it points to an action request, all other kinds of action are assumed except 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

OVO A.02.00 and later

See Also

“OVO Data Structures” on page 445

“OPCDTYPE_MESSAGE” on page 460

“OPCDTYPE_ACTION_REQUEST” on page 447

“OPCDTYPE_ACTION_RESPONSE” on page 449

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 “OVO Data Structures” on page 445 for a list of supported data structures.
data	OVO 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

OVO A.02.00 and later

See Also

“OVO Data Structures” on page 445

opcdata_delete_element()

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

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

Parameters

container	Pointer to an OVO 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

OVO A.03.00 and later

See Also

“OVO Data Structures” on page 445

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

OVO A.02.00 and later

See Also

“OVO Data Structures” on page 445

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 OVO 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 was not valid.

Versions

OVO A.05.00 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	OVO 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

OVO A.07.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 471

“opcdata_get_cmanames()” on page 58

“opcdata_set_cma()” on page 76

“opcdata_remove_cma()” on page 74

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	OVO 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

OVO A.07.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 471

“opcdata_get_cma()” on page 56

“opcdata_set_cma()” on page 76

“opcdata_remove_cma()” on page 74

opcdata_get_double()

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

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

Parameters

`data` OVO 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

OVO A.04.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 471

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	OVO container of type OPCDTYPE_CONTAINER.
element	OVO 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 OVO 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

OVO A.03.00 and later

See Also

“opciter_nth()” on page 87

“OVO Data Structures” on page 445

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

OVO A.05.00 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

OVO A.02.00 and later

See Also

“OVO Data Structures” on page 445

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

Versions

OVO A.02.00 and later

See Also

“OVO Data Structures” on page 445

opcdata_insert_element()

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

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

Parameters

container	OVO data structure of type OPCDTYPE_CONTAINER.
element	OVO data structure.
index	Position in the container where the element is to 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

OVO A.04.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OVO Data Structures” on page 445

opcdata_ladd()

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

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

Parameters

data	Pointer to an OVO 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

OVO A.05.00 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 OVO 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

OVO A.05.00 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 OVO 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 OVO 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

OVO A.05.00 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 OVO 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

OVO A.05.00 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 OVO 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

OVO A.05.00 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 OVO 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

OVO A.05.00 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 OVO 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

OVO A.05.00 and later

opcdata_num_elements()

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

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

Parameters

container	OVO 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

OVO A.04.00 and later

See Also

“OVO Data Structures” on page 445

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

OVO A.07.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 471

“opcdata_get_cma()” on page 56

“opcdata_get_cmanames()” on page 58

“opcdata_set_cma()” on page 76

opcdata_report_error()

CAUTION On Novell NetWare systems, this function only returns the error code. Use the function opcdata_get_error_msg() instead.

```
#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. On Novell NetWare systems, only the error number is returned in text form.

NOTE On Novell NetWare systems, this function is not thread-safe. The thread-safe version of this function is opcdata_get_error_msg().

NOTE The memory allocated by this function must be freed by the caller. This does not apply to Novell NetWare systems.

Return Values

string	string contains error description
	memory must be freed by caller except for Novell NetWare systems

Versions

OVO A.05.00 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

OVO A.07.00 and later

See Also

“OPCDTYPE_MONITOR_MESSAGE” on page 471

“opcdata_get_cma()” on page 56

“opcdata_get_cmanames()” on page 58

“opcdata_remove_cma()” on page 74

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	OVO 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

OVO A.04.00 and later

See Also

“OVO Data Structures” on page 445

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	OVO 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

OVO A.02.00 and later

See Also

“OVO Data Structures” on page 445

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	OVO 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

OVO A.02.00 and later

See Also

“OVO Data Structures” on page 445

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

OVO A.05.00 and later

opciter_begin()

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

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

Parameters

`iterator` OVO 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

If the container is empty, or an error occurred, this function returns NULL.

Versions

OVO A.04.00 and later

See Also

“The OVO Iterator” on page 47

opciter_create()

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

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

Parameters

`iterator` OVO Iterator.

`container` OVO 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

OVO A.04.00 and later

See Also

“The OVO Iterator” on page 47

“OVO Data Structures” on page 445

opciter_end()

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

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

Parameters

`iterator` OVO Iterator.

Description

The function `opciter_end()` returns the pointer to the past-end-element. Because the past-end-element is not defined, 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.

Return Values

`NULL`: past-end-value

Versions

OVO A.04.00 and later

See Also

“The OVO Iterator” on page 47

“OVO Data Structures” on page 445

opciter_free()

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

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

Parameters

`iterator` OVO 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

OVO A.04.00 and later

See Also

“The OVO Iterator” on page 47

“OVO Data Structures” on page 445

opciter_get_pos()

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

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

Parameters

`iterator` OVO 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

OVO A.04.00 and later

See Also

“The OVO Iterator” on page 47

“OVO Data Structures” on page 445

opciter_next()

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

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

Parameters

`iterator` OVO Iterator.

Description

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

Return Values

Reference to the next element or NULL

Versions

OVO A.04.00 and later

See Also

“The OVO Iterator” on page 47

“OVO Data Structures” on page 445

opciter_nth()

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

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

Parameters

`iterator` OVO 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

Reference to the specified element or NULL

Versions

OVO A.04.00 and later

See Also

“opcdata_get_element()” on page 61

“OVO Data Structures” on page 445

opciter_prev()

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

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

Parameters

`iterator` OVO 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--). If the iterator points still to the first container element, or the container is empty, the iterator will be not further decremented and returns NULL.

Return Values

Reference to the previous element or NULL.

Versions

OVO A.04.00 and later

See Also

“OVO Data Structures” on page 445

opciter_set_pos()

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

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

Parameters

`iterator` OVO 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

OVO A.04.00 and later

See Also

“opciter_nth()” on page 87

“OVO Data Structures” on page 445

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. 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:	One of the output parameters is NULL or not of the correct type.

Versions

OVO A.02.00 and later

See Also

“opcreg_create()” on page 91

“opcreg_free()” on page 92

“opcreg_get_long()” on page 93

“opcreg_get_str()” on page 94

“opcreg_set_long()” on page 95

“opcreg_set_str()” on page 96

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. 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:	One of the output parameters is NULL or not of the correct type.
OPC_ERR_NO_MEMORY:	Memory allocation error.

Versions

OVO A.02.00 and later

See Also

“opcreg_free()” on page 92

“opcreg_copy()” on page 90

“opcreg_get_long()” on page 93

“opcreg_get_str()” on page 94

“opcreg_set_long()” on page 95

“opcreg_set_str()” on page 96

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:	One of the output parameters is NULL or not of the correct type.

Versions

OVO A.02.00 and later

See Also

“opcreg_create()” on page 91

“opcreg_copy()” on page 90

“opcreg_get_long()” on page 93

“opcreg_get_str()” on page 94

“opcreg_set_long()” on page 95

“opcreg_set_str()” on page 96

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 in the registration condition.

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

OVO A.02.00 and later

See Also

“opcreg_create()” on page 91

“opcreg_free()” on page 92

“opcreg_copy()” on page 90

“opcreg_get_str()” on page 94

“opcreg_set_long()” on page 95

“opcreg_set_str()” on page 96

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 in the registration condition.

Description

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

Return Values

Returns a pointer to the requested string or if not successful a NULL pointer.

Versions

OVO A.02.00 and later

See Also

“opcreg_create()” on page 91

“opcreg_free()” on page 92

“opcreg_copy()” on page 90

“opcreg_get_long()” on page 93

“opcreg_set_long()” on page 95

“opcreg_set_str()” on page 96

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

The function opcreg_set_long() sets the value of a numeric field of a registration condition.

Return Values

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

Versions

OVO A.02.00 and later

See Also

“opcreg_create()” on page 91

“opcreg_free()” on page 92

“opcreg_copy()” on page 90

“opcreg_get_long()” on page 93

“opcreg_get_str()” on page 94

“opcreg_set_str()” on page 96

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 field of a registration condition.

Return Values

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

Versions

OVO A.02.00 and later

See Also

“opcreg_create()” on page 91

“opcreg_free()” on page 92

“opcreg_copy()” on page 90

“opcreg_get_long()” on page 93

“opcreg_get_str()” on page 94

“opcreg_set_long()” on page 95

Interface API

OVO Interfaces

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

- OVO Message Stream Interface

  This interface makes it possible to read OVO messages from the internal message stream and to write messages into the internal message stream. The Message Stream Interface (MSI) is available on the OVO Management Server and on OVO Managed Nodes. All MSI types establish a connection, either to the OVO message manager for server types (OPCSVIF_*) or to the OVO message agent for agent types (OPCAGTIF_*). This interface is divided into the types:

  - `OPCSVIF_EXTMSGPROC_READ`

    This interface type is used for non-destructive read operations on the OVO 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.

  - `OPCSVIF_EXTMSGPROC_READWRITE`

    This interface type is used to read messages

    from OVO's internal message flow, to modify / create messages, and to write them back to the OVO 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 OVO'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 OVO'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 OVO message flow but is only available on the OVO 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.

- OPCSVIF_EXTAGT_ACTION_REQUEST

  External agents use this interface type to read action requests.

- OPCSVIF_EXTAGT_ACTION_RESPONSE

  Interface instances of this type are used by external agents to write action responses to OVO.

❏ OVO Message Event Interface

This interface allows receipt of OVO 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 OVO Management Server and establishes a connection to the display manager.

- OPCSVIF_MSG_EVENTS

  This interface type is used to receive Message Events caused by changes to messages by other operators.

❏ OVO Application Response Interface

This interface allows receipt of the textual response from previously started application. This interface is available only on the OVO Management Server and establishes a connection to the display manager.

- `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.

Prerequisites

The API functions must be issued by the user root.

Multithread Usage

All functions of the Interface API are safe to be called by multithreaded applications, and are thread-safe for both POSIX Threads and DCE User Threads. They are neither async-cancel, async-signal, nor fork-safe, and cannot be called safely in kernel threads.

`opcreg_copy()` is not thread-safe for POSIX threads or for DCE User Threads.

Registration Conditions of the OVO Interface API

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

OVO provides a set of APIs to create an empty condition, modify or query condition fields and to duplicate or delete a condition definition from memory.

Security Considerations for the OVO 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 OVO 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 OVO message flow.

Automatic / 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 OVO allows access to these message attributes, any user who is authorized to call the APIs is also able to execute actions on OVO managed nodes.

According to the current OVO concept, which regards OVO as an open application providing a high level of flexibility to integrate applications, OVO 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. However, as the checking of a user ID belongs to the OS level with its superuser concept, this conflicts somewhat with the existing OVO concept where the administrator is responsible for the configuration of user roles.

The OVO 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 OVO 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 OVO message flow.

Summary

OVO allows users with a user ID of zero (uid 0), typically root, to access the OVO Interface APIs and to define actions for messages that are sent to the management server. In the GUI, the OVO 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. 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.

Message Events and Application Responses are always discarded if 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

OVO A.02.00 and later

See Also

“OVO Interfaces” on page 97

“Data API” on page 45

“OVO Data Structures” on page 445

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 OVO interface can specify `OPCIF_READ_WAIT` in the opcif_open() 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

OVO A.02.00 and later

See Also

“OVO Interfaces” on page 97

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 (e.g., event correlation engines):

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

Agent Message Stream Interface

Used by external message processors (e.g., event correlation engines):

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

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 (e.g., message browsers)

- `OPCSVIF_MSG_EVENTS`

Application Response Interface

Used by External application processing tools:

- `OPCSVIF_APPLIC_RESPONSE`

`instance` Name of the interface instance which is registered for one of the interface types above. The name is limited to a length of 12 alpha-numeric characters because it is also part of the queue filename.

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

- `OPCIF_ALWAYS` (default)
- `OPCIF_SV_RUNNING`
- `OPCIF_AGT_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)
- `OPCIF_READ_NOWAIT`

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)
- `OPCIF_CLOSE_DISCARD`

In the first case, messages in the read-queue are appended to the write-queue; in the second, these messages are discarded. This option applies only for the `OPCSVIF_EXTMSGPROC_READWRITE` interface type.

- `OPCIF_IGNORE_MSI_ALREADY_EXISTS`

  This flag allow you to reuse an existing queue file.

  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 '|' operator.

`max_entries` Specifies the maximum number of entries in the read-queues. If this number is exceeded, OVO 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

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

OVO A.02.00 and later

See Also

“OVO Interfaces” on page 97

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	OVO 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 has been opened with OPCIF_READ_WAIT, the calling process is blocked until the information is available. 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
OPC_ERR_CANT_READ_MSG:	reading message, event, or response failed
OPC_ERR_NO_MEMORY:	memory allocation failed

Versions

OVO A.02.00 and later

See Also

“OVO Interfaces” on page 97

“OVO Data Structures” on page 445

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

- Application Responses

  Defines a certain application response specified by the application response ID

`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 (e.g., 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 489.

❏ Message Attributes

OVO supports registration for message type, message group, node name, object, severity and application attributes. You can also combine attributes (logical AND), and ‘|’ within an attribute (logical OR). Multiple registrations (logical OR 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:

OVO supports the registration for an event mask and the restriction of events to certain operators. Multiple registrations for the operator restriction (logical OR 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:

OVO 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

OVO A.02.00 and later

See Also

“OVO Interfaces” on page 97

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 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

OVO A.02.00 and later

See Also

“OVO Interfaces” on page 97

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` OVO data structure.

Description

Use the function `opcif_write()` to write a message to the OVO Interface.

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

OVO A.02.00 and later

See Also

“OVO Interfaces” on page 97

“OVO Data Structures” on page 445

opcreg_copy()

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

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

Parameters

reg_cond	OVO registration condition structure.
copy	Copy of reg_cond.

Description

The API creates a copy of the condition and returns it in copy. The allocated memory has to be deallocated using opcreg_free(). copy must be freed using opcreg_free() before calling this function.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_OUTPARAM:	copy is invalid
OPC_ERR_NO_MEMORY:	allocation of memory failed

Versions

OVO A.02.00 and later

See Also

“opcreg_create()” on page 117

“opcreg_free()” on page 118

“opcreg_get_long()” on page 119

“opcreg_get_str()” on page 120

“opcreg_set_long()” on page 121

“opcreg_set_str()” on page 122

opcreg_create()

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

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

Parameters

reg_cond OVO registration condition structure.

Description

This routine creates an empty registration condition. 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

OVO A.02.00 and later

See Also

“opcreg_copy()” on page 116

“opcreg_free()” on page 118

“opcreg_get_long()” on page 119

“opcreg_get_str()” on page 120

“opcreg_set_long()” on page 121

“opcreg_set_str()” on page 122

opcreg_free()

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

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

Parameters

reg_cond OVO registration condition structure.

Description

The function opcreg_free() deallocates memory previously allocated by opcreg_create(), opcreg_copy(), or opcreg_set_…().

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_OUTPARAM:	reg_cond is invalid

Versions

OVO A.02.00 and later

See Also

“opcreg_copy()” on page 116

“opcreg_create()” on page 117

“opcreg_get_long()” on page 119

“opcreg_get_str()” on page 120

“opcreg_set_long()” on page 121

“opcreg_set_str()” on page 122

opcreg_get_long()

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

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

Parameters

reg_cond	OVO registration condition structure.
field	Selects 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

OVO A.02.00 and later

See Also

“opcreg_copy()” on page 116

“opcreg_create()” on page 117

“opcreg_free()” on page 118

“opcreg_get_str()” on page 120

“opcreg_set_long()” on page 121

“opcreg_set_str()” on page 122

opcreg_get_str()

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

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

Parameters

reg_cond	OVO registration condition structure.
attribute	Selects the attribute that is queried.

Description

Use the routine opcdata_get_str() to access the attribute values of a 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.

Versions

OVO A.02.00 and later

See Also

“opcreg_copy()” on page 116

“opcreg_create()” on page 117

“opcreg_free()” on page 118

“opcreg_get_long()” on page 119

“opcreg_set_long()” on page 121

“opcreg_set_str()” on page 122

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	OVO registration condition structure.
field	Select the attribute that is set.
value	Specify the value of the attribute.

Description

Use the function opcreg_set_long() to set attributes to a certain value.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_FIELD	field is invalid

Versions

OVO A.02.00 and later

See Also

“opcreg_copy()” on page 116

“opcreg_create()” on page 117

“opcreg_free()” on page 118

“opcreg_get_long()” on page 119

“opcreg_get_str()” on page 120

“opcreg_set_str()” on page 122

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	OVO registration condition structure.
field	Select the attribute that is set.
value	Specify the value of the attribute.

Description

Use the function opcreg_set_str() to set attributes to a certain value.

Return Values

OPC_ERR_OK	OK
OPC_ERR_INVALID_FIELD	field is invalid

Versions

OVO A.02.00 and later

See Also

“opcreg_copy()” on page 116

“opcreg_create()” on page 117

“opcreg_free()” on page 118

“opcreg_get_long()” on page 119

“opcreg_get_str()” on page 120

“opcreg_set_long()” on page 121

Server Message API

OVO provides a set of functions to access and modify a message and its details, get, add, or delete annotations, and start actions by way of the API.

Data Structures

OPCDTYPE_MESSAGE

OPCDTYPE_MESSAGE_ID

OPCDTYPE_ANNOTATION

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 OVO database as an OVO user or administrator, restricting the access privileges according to the restrictions of the given operator.

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

`/var/opt/OV/log/OpC/mgmt_sv/opcerror`

Prerequisites

The functions are only available on the OVO management server.

Multithread Usage

All functions of the Server Message API are safe to be called by multithreaded applications, and are thread-safe for both POSIX Threads and DCE User Threads. They are neither async-cancel, async-signal, nor fork-safe, and cannot be safely called in kernel threads.

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 OVO 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.

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:	message 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 UI 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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE” on page 460

“OPCDTYPE_MESSAGE_ID” on page 470

“OPCDTYPE_ANNOTATION” on page 451

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 OVO 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.

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:	message 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 UI 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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE” on page 460

“OPCDTYPE_MESSAGE_ID” on page 470

“OPCDTYPE_ANNOTATION” on page 451

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 OVO 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:	message ID is malformed or database contains no message with that ID
OPC_ERR_DATABASE_ERROR:	operator/message check failed; information retrieval for UI 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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE” on page 460

“OPCDTYPE_MESSAGE_ID” on page 470

“OPCDTYPE_ANNOTATION” on page 451

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 OVO 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.

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:	message 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 UI 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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE” on page 460

“OPCDTYPE_MESSAGE_ID” on page 470

“OPCDTYPE_ANNOTATION” on page 451

opcmsg_ack()

```
#include opcsvapi.h

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

Parameters

opc_conn	Internal OVO connection information.
message_id	Message ID of the message to be acknowledged.

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 DB information for UI 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 UI
OPC_ERR_ALREADY_DONE:	message is already acknowledged

OPC_ERR_MSG_OWNED_BY_ANOTHER_USER: message is owned by another OVO user

Versions

OVO A.03.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

opcmsg_disown()

```
#include opcsvapi.h

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

Parameters

opc_conn	Internal OVO connection information.
message_id	Message ID of the message to be disowned.

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 DB information for UI 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 UI information request
OPC_ERR_CANT_INFORM_UI:	cannot inform UI

OPC_ERR_MSG_OWNED_BY_ANOTHER_USER: message is owned by another OVO user

Versions

OVO A.04.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

opcmsg_escalate()

```
#include opcsvapi.h

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

Parameters

opc_conn	Internal OVO connection information.
message_id	Message ID of the message to be escalated.

Description

Use the function opcmsg_escalate() to escalate a message to the responsible escalation server. The escalation management server is specified by the administrator and is part of the configuration of the management server.

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_ESCALATED.

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 node_info, cannot get DB information for UI change request
OPC_ERR_ACCESS_DENIED:	operator is not allowed to escalate the message
OPC_ERR_ESCALATION_FAILED:	escalation failed

OPC_ERR_NO_ESCALATION_DEFINED: no escalation server defined

OPC_ERR_CANT_INFORM_UI: cannot inform UI

OPC_ERR_MSG_OWNED_BY_ANOTHER_USER: message is owned by another OVO user

OPC_ERR_ACTION_RUNNING: message has a running action

OPC_ERR_MSG_NOT_ACKNOWLEDGED: cannot acknowledge message after escalation

OPC_ERR_INVAL_NODE: escalation manager not in node bank

Versions

OVO A.04.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

opcmsg_get()

```
#include opcsvapi.h

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

Parameters

opc_conn	Connection information retrieved by opc_connect().
message_id	ID of the message information should be obtained for.
message	OVO data structure.

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

OVO A.03.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

“OPCDTYPE_MESSAGE” on page 460

opcmsg_get_instructions()

```
#include opcsvapi.h

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

Parameters

opc_conn	OVO connection information retrieved by opc_connect().
message_id	OVO message ID structure.

Description

Use the function `opcmsg_get_instructions()` to get instructions for a message.

Return Values

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

Versions

OVO A.03.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

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 OVO database.
msg_id	Message ID of the message to be selected: must be of type OPCDTYPE_MESSAGE_ID or OPCDTYPE_MESSAGE.
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. Changes to other message attributes are not possible; they are ignored by OVO.

Each successful modification is documented by an annotation that includes the name of the OVO 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 OVO 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

“OPCDTYPE_MESSAGE” on page 460

opcmsg_own()

```
#include opcsvapi.h

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

Parameters

opc_conn	Internal OVO connection information.
message_id	Message ID of the message to be owned.

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 DB information for UI 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 UI information request
OPC_ERR_CANT_INFORM_UI:	cannot inform UI

OPC_ERR_MSG_OWNED_BY_ANOTHER_USER: message is owned by another OVO user

Versions

OVO A.03.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

“OPCDTYPE_MESSAGE” on page 460

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 OVO database.
msg_id	Message ID of the message to be selected: must be of type OPCDTYPE_MESSAGE_ID or OPCDTYPE_MESSAGE.
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 OVO 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 OVO 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 UI of the select event

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

“OPCDTYPE_MESSAGE” on page 460

opcmsg_start_auto_action()

```
#include opcsvapi.h

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

Parameters

opc_conn	Connect to the OVO database.
message_id	ID of the message the action should be started for.

Description

Use the function opcmsg_start_auto_action() to restart an automatic 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_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 DB information for UI 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 UI information request
OPC_ERR_CANT_INFORM_UI:	cannot inform UI about change
OPC_ERR_MSG_OWNED_BY_ANOTHER_USER:	message is owned by another OVO user
OPC_ERR_ACTION_RUNNING:	message has a running action

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

opcmsg_start_op_action()

```
#include opcsvapi.h

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

Parameters

opc_conn	Connect to the OVO database.
message_id	ID of the message the action should be started for.

Description

Use the function `opcmsg_start_op_action()` to start an operator-initiated action if one is defined for the given message. This function informs all running GUIs.

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 DB information for UI 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 UI information request.
OPC_ERR_CANT_INFORM_UI:	cannot inform UI about change
OPC_ERR_MSG_OWNED_BY_ANOTHER_USER:	message is owned by another OVO user
OPC_ERR_ACTION_RUNNING:	message has a running action

Versions

OVO A.04.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

opcmsg_unack()

```
#include opcsvapi.h

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

Parameters

opc_conn	Internal OVO connection information.
message_id	Message ID of the message to be unacknowledged.

Description

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

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 DB information for UI 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 UI information request
OPC_ERR_CANT_INFORM_UI:	cannot inform UI

OPC_ERR_ALREADY_DONE: message is already unacknowledged

Versions

OVO A.04.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“OPCDTYPE_MESSAGE_ID” on page 470

Agent Message API

OVO 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 160 for functions to send monitor values.

Data Structures

OPCDTYPE_MESSAGE_ID

OPCDTYPE_MESSAGE

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 opc_op.

Each opdata 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 both POSIX Threads and DCE User Threads. They are neither async-cancel, async-signal, nor fork-safe, and cannot be safely called in kernel threads.

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 in the file `/opt/OV/bin/OpC/install/opcinfo` using the parameter `OPC_STORE_TIME_FOR_MGR_INFO`. The specified value is the time in hours with a default setting of one hour if this parameter is not changed.

```
...
OPC_TRACE                           TRUE
OPC_STORE_TIME_FOR_MGR_INFO         2
...
```

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 user opc_op or root. If not, the customer 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

OVO A.04.00 and later

See Also

“opcagtmsg_send()” on page 156

“opcmsg()” on page 158

“OPCDTYPE_MESSAGE_ID” on page 470

opcagtmsg_send()

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

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

Parameters

message — Message of type OPCDTYPE_MESSAGE.

Description

Use the function opcagtmsg_send() to send a message, created on the managed node, to its responsible manager. The message must be of type OPCDTYPE_MESSAGE. The message ID can be retrieved from the message object using opcdata_get_str() immediately after the send call was executed.

Only the message attributes Severity, Application, Message Group, Object, Message Text, Option Strings and Node are used in opcagtmsg_send().

The API program must run as user opc_op or root (if not, customer program has to setuid).

After opcagtmsg_send() was called it is possible to get the ID of the sent message using:

opcdata_get_str(message, OPCDATA_MSGID)

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

OVO A.04.00 and later

See Also

“opcagtmsg_ack()” on page 155

“opcmsg()” on page 158

“OPCDTYPE_MESSAGE” on page 460

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.
application	Application of the message source.
object	Object of the message source.
msg_text	Message text.
msg_group	Message group.
nodename	Name of the node originating the message.

Description

Use the function opcmsg() to send a message, created on the managed node, to the management server. 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

OVO A.02.00 and later

See Also

“opcagtmsg_ack()” on page 155

“opcagtmsg_send()” on page 156

Agent Monitor API

OVO 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 users root or opc_op.

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

Multithread Usage

All functions of the Agent Monitor API are safe to be called by multithreaded applications, and are thread-safe for both POSIX Threads and DCE User Threads. They are neither async-cancel, async-signal, nor fork-safe, and cannot be safely called in kernel threads.

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.

Description

Use the function opcagtmon_send( ) to send a monitor value, created on the managed node, to its responsible manager. The message must be of type OPCDTYPE_MONITOR_MESSAGE.

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

The API program must be run as user opc_op or root. If not, the customer 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

OVO A.04.00 and later

See Also

“opcmon()” on page 162

“OPCDTYPE_MONITOR_MESSAGE” on page 471

opcmon()

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

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

Parameters

objname	Name of the monitored object.
monval	Actual value of the monitored object.

Description

Use the function opcmon( ) to send a monitor value, created on the managed node, to its responsible management server.

The API program must run as user opc_op or root. If not, the customer 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

OVO A.02.00 and later

See Also

“opcagtmon_send()” on page 161

3 Functions of the OVO Configuration APIs

In This Chapter

The **OVO Configuration API** provides functions which allow you to access the OVO 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
- Template 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/`.

Configuration API Usage

The functions opc*_modify() always modify *all* attributes of the OVO 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 can *not* be used to specify object attributes for a modify operation.

Example of an Object Modification

The following example modifies the label of an OVO 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

Data Structures

opc_connection

Usage

The Connection API can be issued by any user.

Prerequisites

The connection API is only available on the management server.

Multithread Usage

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

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 OVO database, to 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 OVO GUI. Several programs can connect to OVO 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

OVO A.03.00 and later

See Also

“opc_disconnect()” on page 170

“opcconn_get_capability()” on page 171

“opcconn_set_capability()” on page 173

opc_disconnect()

```
#include opcsvapi.h

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

Parameters

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

Description

Use the function opc_disconnect() to disconnect from the OVO database and to free the memory used for the connection information. This has to 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

OVO A.03.00 and later

See Also

“opc_connect()” on page 168

“opcconn_get_capability()” on page 171

“opcconn_set_capability()” on page 173

opcconn_get_capability()

```
   #include opcsvapi.h

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

Parameters

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

attrib The following attributes are accessible with these functions:

- OPCDATA_DM_CONNECTED

  TRUE or FALSE, contains the connection status to the OVO display manager. When the display manager was running during the connection process, TRUE will be returned.

- OPCDATA_ACKNOWLEDGE

  TRUE or FALSE, returns whether the operator is able to acknowledge messages.

- OPCDATA_PERFORM_ACTION

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

- OPCDATA_CHANGE_MSG

  TRUE or FALSE, returns whether the operator is able to change message attributes.

- OPCDATA_OWN

  TRUE or FALSE, returns whether the operator is able to own/disown messages.

- OPCDATA_IMMEDIATE_SYNC

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

value Returned value for the requested attribute.

Description

Allows to access the capabilities of the connected OVO operator.

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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“opcconn_set_capability()” on page 173

opcconn_set_capability()

```
   #include opcsvapi.h

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

Parameters

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

attrib The following attributes can be set with these functions:

- OPCDATA_DM_CONNECTED

  TRUE or FALSE, contains the connection status to the OVO display manager. When the display manager was running during the connection process, TRUE will be returned.

- OPCDATA_ACKNOWLEDGE

  TRUE or FALSE, returns whether the operator is able to acknowledge messages.

- OPCDATA_PERFORM_ACTION

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

- OPCDATA_CHANGE_MSG

  TRUE or FALSE, returns whether the operator is able to change message attributes.

- OPCDATA_OWN

  TRUE or FALSE, returns whether the operator is able to own/disown messages.

- OPCDATA_IMMEDIATE_SYNC

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

value Returned value for the requested attribute.

Description

Allows to set some capabilities of the connection.

Return Values

OPC_ERR_INVALID_INPARAM:	opc_conn, attrib, or 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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“opcconn_get_capability()” on page 171

Application Configuration API

The application API provides a set of functions to configure OVO applications. To use these functions, except for `opcappl_start()`, it is necessary to connect to the management server as administrator (see “opc_connect()” on page 168). `opcappl_start()` can be called with a connection of any valid user.

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

Error information is written to the error logfile `/var/opt/OV/log/OpC/mgmt_sv/opcerror` 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 52 or “opcdata_clear()” on page 49) and freeing (see “opcdata_free()” on page 54) the needed memory.

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect.

Data Structures

`OPCDTYPE_APPL_CONFIG`

Usage

The Application Configuration API can be called by any user.

Prerequisites

The Application Configuration API is only available on the management server.

Multithread Usage

All functions of the OVO Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads, DCE User 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 OVO 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 OVO 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 ID of the application is returned in appl. Note that this function does not add OV Applications or OV Services.

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 191 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

“opcapplgrp_assign_appls()” on page 191

“opcdata_create()” on page 52

opcappl_delete()

```
#include opcsvapi.h

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

Parameters

opc_conn	Connection to the OVO database
appl	Application configuration of type OPCDTYPE_APPL_CONFIG

Description

Deletes the specified application. The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“opc_connect()” on page 168

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 OVO 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.

The appl and appl_conf opcdata must be of 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“opc_connect()” on page 168

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 OVO 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_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Allocation of memory failed.
OPC_ERR_OBJECT_NOT_FOUND	Application bank not found.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_CONTAINER” on page 446

“opc_connect()” on page 168

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 OVO 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.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“opc_connect()” on page 168

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	Contains information about the connection
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 ID to identify application for the Application Response Interface

Description

Sends application execution requests from the specified application to the nodes in node_list. When the request is sent successfully, “opcappl_start()” on page 184 returns OPC_ERR_OK. 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 ID app_exec_id is a regular string and can contain up to 36 characters. This ID is needed to receive the application response from the OVO Application Response Interface (see also opcif_api(3)). The function is not able to return the application response directly.

Return Values

OPC_ERR_OK	Execution request successfully sent.
OPC_ERR_INVALID_INPARAM	opc_conn, application or node_list is NULL. appl_exec_id is NULL or has zero length.
OPC_ERR_INVALID_OPCDATA_TYPE	application or node_list 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_NO_MEMORY	Allocation of memory failed.
OPC_ERR_INVALID_APPLICATION	Application not found in the database.
OPC_ERR_INVALID_APP_PARAM	Application parameter contain ';'.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_NODE” on page 472

“opc_connect()” on page 168

Application Group Configuration API

The application group API provides a set of functions to configure OVO application groups. To use these functions, it is necessary to connect to the management server as administrator, (see opc_connect(3)).

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

Error information is written to the error logfile /var/opt/OV/log/OpC/mgmt_sv/opcerror 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.

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect.

Data Structures

OPCDTYPE_APPL_CONF

Usage

The Application Group Configuration API can be called by any user.

Prerequisites

The Application Group Configuration API is only available on the management server.

Multithread Usage

All functions of the OVO Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads, DCE User 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 OVO 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 ID 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	appl is NULL or invalid type.
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_OBJECT_ALREADY_EXISTS	Application group name already exists.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

“opcdata_create()” on page 52

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 OVO 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 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 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	appl_list is NULL or invalid.
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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

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 OVO 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 same application can have more than one parent application groups.

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 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

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 OVO 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.

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 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

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 OVO 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.

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 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

opcapplgrp_delete()

```
#include opcsvapi.h

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

Parameters

opc_conn	Connection to the OVO 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

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 OVO 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

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 OVO 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 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 top level applications are returned.

The parameter appl_list must be an “OPCDTYPE_CONTAINER” on page 446 of the type OPCDTYPE_EMPTY or “OPCDTYPE_APPL_GROUP” on page 455.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

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 OVO 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 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 top level 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_CONTAINER” on page 446

“opc_connect()” on page 168

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 OVO 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 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_CONTAINER” on page 446

“opc_connect()” on page 168

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 OVO 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.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“opc_connect()” on page 168

Message Group Configuration API

The Message Group Configuration API provides a set of functions to configure OVO message groups. To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect, see page 168.

Data Structures

OPCDTYPE_MESSAGE_GROUP

Usage

The Message Group Configuration API can be called by any user.

Prerequisites

The Message Group Configuration API is only available on the management server.

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, DCE User 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	Connect to the OVO database
msg_group	opcdata structure of type OPCDTYPE_MESSAGE_GROUP

Description

Use the function opcmsggrp_add() to add a message group to the OVO database.

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 > 0. 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_MESSAGE_GROUP” on page 469

opcmsggrp_delete()

```
#include opcsvapi.h

int opcmsggrp_delete (
        const opc_connection  opc_conn,          /* in */
        const opcdata         msg_group          /* in */
                        );
```

Parameters

opc_conn	Connect to the OVO database
msg_group	opcdata structure of type OPCDTYPE_MESSAGE_GROUP

Description

Use the function opcmsggrp_delete() to remove a message group from the OVO database.

NOTE Deleting a message group also changes the configuration of the operator who is responsible for that message group. The operator must restart the GUI for the changes to take effect.

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_INVALID_MESSAGE GROUP:	message group is not in the database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_MESSAGE_GROUP” on page 469

opcmsggrp_get_list()

```
#include opcsvapi.h

int opcmsggrp_get_list (
        const opc_connection   opc_conn,         /* in */
        opcdata                msg_groups        /* out */
                        );
```

Parameters

opc_conn	Connect to the OVO database
msg_groups	Container of type OPCDTYPE_MESSAGE_GROUP

Description

Use the function opcmsggrp_get_list() to get a list of all message groups. A message group is identified by its name.

If the function is called for an OVO user other than the administrator the message groups of this OVO user 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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_MESSAGE_GROUP” on page 469

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	Connect to the OVO database
msg_group	opcdata 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 of a message group.

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_INVALID_MESSAGE GROUP:	message group is not in the database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_MESSAGE_GROUP” on page 469

Message Regroup Condition Configuration API

The message regroup condition API provides a set of functions to configure OVO message regroup conditions. To use these functions, it is necessary to connect to the management server as administrator, (see opc_connect(3)).

A message regroup condition is specified either by name or by the uuid. If the uuid is given, a specified name will be ignored.

Error information is written to the error logfile /var/opt/OV/log/OpC/mgmt_sv/opcerror 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.

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect.

Data Structures

OPCDTYPE_REGROUP_COND

Usage

The Message Regroup Condition Configuration API can be called by any user.

Prerequisites

The Message Regroup Condition Configuration API is only available on the management server.

Multithread Usage

All functions of the OVO Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads, DCE User 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	Contains information about the connection
mcond	Message regroup conditions to add

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.

The mcond opcdata must be of type OPCDTYPE_REGROUP_COND.

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_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Condition already exists.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 485

“opc_connect()” on page 168

“opcdata_create()” on page 52

opcmsgregrp_delete()

```
#include opcsvapi.h

int opcmsgregrp_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        mcond     /* in/out */
      );
```

Parameters

opc_conn	Contains information about the connection
mcond	Message regroup condition to be deleted

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.

The mcond opcdata must be of type OPCDTYPE_REGROUP_COND.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 485

“opc_connect()” on page 168

opcmsgregrp_get()

```
#include opcsvapi.h

int opcmsgregrp_get (
      opc_connection opc_conn,   /* in/out */
      opcdata        mcond,      /*     in */
      opcdata        mcond_conf  /*    out */
      );
```

Parameters

opc_conn	Contains information about the connection
mcond	Message regroup condition to be returned
mcond_conf	Configuration of matching condition

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.

The mcond and mcond_conf opcdata must be of type OPCDTYPE_REGROUP_COND.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 485

“opc_connect()” on page 168

opcmsgregrp_get_list()

```
#include opcsvapi.h

int opcmsgregrp_get_list (
      opc_connection opc_conn,   /* in/out */
      opcdata        mcond_list  /*    out */
      );
```

Parameters

opc_conn	Contains information about the connection
mcond_list	'Ordered' list of all conditions will be returned. mcond_list must be of type OPCDTYPE_CONTAINER

Description

Gets a list of all message regroup conditions with the full configuration of each single message regroup condition.

The parameter mcond_list must be an OPCDTYPE_CONTAINER of type OPCDTYPE_EMPTY or OPCDTYPE_REGROUP_COND.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_REGROUP_COND” on page 485

“opc_connect()” on page 168

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	Contains information about the connection
mcond	Message regroup condition to be modified
mcond_conf	Modified condition data

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.

The mcond and mcond_conf opcdata must be of type OPCDTYPE_REGROUP_COND.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 485

“opc_connect()” on page 168

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	Contains information about the connection
mcond	Message regroup condition to be moved
to_pos	New position in condition list. The list is '1' based the new position must be between 1 and LAST_ELEMENT other positions generate an error

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.

The mcond opcdata must by of type OPCDTYPE_REGROUP_COND.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_REGROUP_COND” on page 485

“opc_connect()” on page 168

Node Configuration API

The Node Configuration API provides a set of functions to configure OVO managed nodes and OVO node groups. In addition, it allows to add, modify, and delete managed nodes and node groups, and to change the attributes of a node.

Data Structures

The Node Configuration API works with opcdata structures of type OPCDTYPE_NODE and OPCDTYPE_NODE_CONFIG. The OPCDTYPE_NODE structure contains a subset of the OPCDTYPE_NODE_CONFIG structure; it contains only the most important information to specify a node. It is mainly used to get an overview of all nodes, or to specify a node for modification.

OPCDTYPE_NODE	Contains a minimal set of node information, for example, IP address, network type and machine type.
OPCDTYPE_NODE_CONFIG	Contains the full set of OVO node attributes

For an overview of all node configuration attributes, see Table 5-16, “OPCDTYPE_NODE_CONFIG,” on page 474.

Usage

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

To use these functions, it is necessary to connect to the management server as administrator by running opc_connect(). This does not apply to the function opcnode_get_list().

Prerequisites

The functions can only be called by an OVO user with administrator rights. Only the function opcnode_get_list()can be called by any other user.

Multithread Usage

All functions of the OVO Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads, DCE User 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	Connect to the OVO database
node_conf	Node configuration of type OPCDTYPE_NODE_CONFIG

Description

Use the function opcnode_add() to add a given managed node to the OVO 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.

A node is identified by its name or its ID; the ID has higher priority.

You can fill the node configuration structure with data using the opcdata API, see “Data API” on page 45. 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 241 for more information.

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_OBJECT_ALREADY_EXISTS: node is already in the database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opcnode_get_defaults()” on page 241

“opcnodegrp_assign_nodes()” on page 250

“OPCDTYPE_NODE_CONFIG” on page 474

opcnode_assign_templates()

```
#include opcsvapi.h

int opcnode_assign_templates (
        const opc_connection   opc_conn,         /* in */
        opcdata                mgd_node,         /* in */
        opcdata                templates         /* in */
                              );
```

Parameters:

opc_conn	Connect to the OVO database
mgd_node	Name or IP address of the node; could be of type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG
templates	Description of OVO template(s); container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opccnode_assign_templates() to assign templates to a managed node. Each template must be of the type OPCDTYPE_TEMPLATE_INFO, and an opcdata container of the same type must be used. If a specified template is already assigned to the node the assignment will be ignored and a warning will be returned. The OPCDATA_STATUS field of the template information will be set to OPC_ERR_ALREADY_DONE. Note that the function only works with opcdata templates and not with template files.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL managed_node is NULL template_list 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:	managed node is not in the database

OPC_ERR_NOT_COMPLETELY_ DONE:	not all templates were assigned to the node
OPC_WARN_EMPTY_CONTAINER:	no templates were specified in templates

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

opcnode_deassign_templates()

```
#include opcsvapi.h

int opcnode_deassign_templates (
        const opc_connection   opc_conn,         /* in */
        opcdata                mgd_node,         /* in */
        opcdata                templates         /* in */
                                );
```

Parameters

opc_conn	Connect to the OVO database
mgd_node	Name or IP address of the node; could be of type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG
templates	Description of OVO template(s); this is a container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opccnode_deassign_templates() to deassign templates from a managed node. Each template must be of the type OPCDTYPE_TEMPLATE_INFO, and an opcdata container of the same type must be used. If a specified template has already been deassigned from the node the deassignment will be ignored and a warning will be returned. The OPCDATA_STATUS field of the template information will be set to OPC_ERR_ALREADY_DONE. Note that the function only works with opcdata templates and not with template files.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL managed_node is NULL template_list 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:	managed node is not in the database

OPC_ERR_NOT_COMPLETELY_DONE:	not all templates were deassigned
OPC_WARN_EMPTY_CONTAINER:	no templates were specified in templates

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

opcnode_delete()

```
#include opcsvapi.h

int opcnode_delete (
         const opc_connection   opc_conn,          /* in */
         const opcdata          node               /* in */
                  );
```

Parameters

opc_conn	Connect to the OVO database
node	Name or IP address of the node to be deleted; could be of type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG

Description

opccnode_delete()deletes the specified node from the OVO 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 template, application, or message, a warning is issued. Generate the Node Reference Report to identify all references.

NOTE Deleting a node from the OVO 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 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_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 templates, applications, or messages

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

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	Connect to the OVO 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 OVO database. The returned data structure of the type OPCDTYPE_NODE_CONFIG contains all necessary information to define a node in OVO. 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_INVALID_NODE:	managed node is not in the database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

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	Connect to the OVO database
mgd_node	Node of type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG
node_conf	Node configuration of type OPCDTYPE_NODE_CONFIG

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. 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. You 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_INVALID_NODE_TYPE:	invalid network machine type

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opcdata_create()” on page 52

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

opcnode_get_list()

```
#include opcsvapi.h

int opcnode_get_list (
         const opc_connection   opc_conn,          /* in */
         opcdata                nodes              /* out */
                     );
```

Parameters

opc_conn	Connect to the OVO database
nodes	Container of type OPCDTYPE_EMPTY or OPCDTYPE_NODE

Prototype

Use the function opcnode_get_list() to get a list of all nodes from the OVO database. 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_DATABASE_ERROR:	access to database failed
OPC_ERR_NO_MEMORY:	out of memory

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE” on page 472

opcnode_get_templates()

```
#include opcsvapi.h

int opcnode_get_templates (
         const opc_connection   opc_conn,          /* in */
         const opcdata          mgd_node,          /* in */
         opcdata                templates          /* out */
                          );
```

Parameters

opc_conn	Connect to the OVO database
mgd_node	Name or IP address of the node; could be of type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG
templates	container of type OPCDTYPE_TEMPLATE_INFO

Description

opccnode_get_templates() returns a list of all templates and template groups that are assigned to the managed node. The template list is returned in a container of the type OPCDTYPE_TEMPLATE_INFO. This container element does not contain a complete definition of the template, only the template name, template type and template ID. You can use these elements and the template file API to get a full description of the template. The container can also contain template groups which can be identified by the template type.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL mgd_node is NULL templates 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:	managed node is not in the database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

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	Connect to the OVO database
mgd_node	Name or IP address of the node to be modified; could be of type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG
new_node_conf	New node configuration of type OPCDTYPE_NODE_CONFIG

Description

Use the function opcnode_modify() to change the attributes of an already existing node in the OVO database. The modifications are done in the node configuration structure using the opcdata API, see “Data API” on page 45.

To retrieve a list of managed nodes, use the function opcnode_get_list(), see “opcnode_get_list()” on page 243. 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_INVALID_NODE:	managed node is not in the database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

opcnodegrp_add()

```
#include opcsvapi.h

int opcnodegrp_add (
        const opc_connection   opc_conn,         /* in */
        opcdata                node_group        /* in */
                    );
```

Parameters:

opc_conn	Connect to the OVO 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 OVO database. 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 250 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opcnodegrp_assign_nodes()” on page 250

“OPCDTYPE_NODE_GROUP” on page 483

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	Connect to the OVO database
node_group	Node group to be assigned; opcdata structure of type OPCDTYPE_NODE_GROUP
nodes	opcdata container of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG

Description

Use the function opcnodegrp_assign_nodes() to assign nodes to a node group. 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 operator must restart the GUI for the changes to take effect.

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_NOT_COMPLETELY_DONE:	not all nodes were assigned to the node group
OPC_WARN_EMPTY_CONTAINER:	no nodes were specified in nodes

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

opcnodegrp_assign_templates()

```
#include opcsvapi.h

int opcnodegrp_assign_templates (
        const opc_connection   opc_conn,         /* in */
        const opcdata          node_group,       /* in */
        const opcdata          templates         /* in */
                                 );
```

Parameters

opc_conn	Connect to the OVO database
node_group	Node group to which the templates are to be assigned; opcdata structure of type OPCDTYPE_NODE_GROUP
templates	opcdata container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opcnodegrp_assign_templates() to assign templates to a node group.

If a template 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 templates 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_NOT_COMPLETELY_DONE:	not all templates were assigned to the node (group)
OPC_WARN_EMPTY_CONTAINER:	no templates were specified in templates

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

“OPCDTYPE_TEMPLATE_INFO” on page 486

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	Connect to the OVO database
node_group	Node group from which the nodes are to be deassigned; opcdata structure of type OPCDTYPE_NODE_GROUP
nodes	opcdata container of the type OPCDTYPE_NODE or OPCDTYPE_NODE_CONFIG

Description

Use the function opcnodegrp_deassign_nodes() to deassign nodes from a node group. 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 operator must restart the GUI for the changes to take effect.

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_NOT_COMPLETELY_DONE:	not all nodes were deassigned from the node group
OPC_WARN_EMPTY_CONTAINER:	no nodes were specified in nodes

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODE_CONFIG” on page 474

opcnodegrp_deassign_templates()

```
#include opcsvapi.h

int opcnodegrp_deassign_templates (
       const opc_connection   opc_conn,         /* in */
       const opcdata          node_group,       /* in */
       const opcdata          templates         /* in */
                                    );
```

Parameters

opc_conn	Connect to the OVO database
node_group	Node group from which the templates are to be deassigned; opcdata structure of type OPCDTYPE_NODE_GROUP
templates	opcdata container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opcnodegrp_deassign_templates() to deassign templates from a node group.

If a template 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 templates 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_NOT_COMPLETELY_DONE:	not all templates were deassigned from the node group

OPC_WARN_EMPTY_CONTAINER:	no templates were specified in templates

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

“OPCDTYPE_TEMPLATE_INFO” on page 486

opcnodegrp_delete()

```
#include opcsvapi.h

int opcnodegrp_delete (
        const opc_connection   opc_conn,         /* in */
        const opcdata          node_group        /* in */
                        );
```

Parameters

opc_conn	Connect to the OVO 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 OVO database and remove all node assignments.

NOTE Deleting a node group from the OVO database also changes the configuration of the operators who are responsible for that node. The operator must restart the GUI for the changes to take effect.

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_INVALID_NODE_GROUP:	node group is not in the database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

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	Connect to the OVO database
node_group	Node group specification of the type OPCDTYPE_NODE_GROUP
nodegrp_conf	opcdata 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 ID 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

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 OVO database
container	opcdata 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 250 and “opcnodegrp_deassign_nodes()” on page 254 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 264 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

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	Connect to the OVO database
node_group	Node group in opcdata structure of type OPCDTYPE_NODE_GROUP
nodes	opcdata 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. 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 229 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

“OPCDTYPE_NODE” on page 472

opcnodegrp_get_templates()

```
#include opcsvapi.h

int opcnodegrp_get_templates (
       const opc_connection   opc_conn,         /* in */
       const opcdata          node_group,       /* in */
       const opcdata          templates         /* out */
                              );
```

Parameters

opc_conn	Connect to the OVO database
node_group	opcdata structure of type OPCDTYPE_NODE_GROUP
templates	opcdata container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opcnodegrp_get_templates() to get a list of the templates and template groups assigned to a node group. This list could be used to change the template to node group assignments using the functions opcnodegroup_assign_templates() and opcnodegroup_deassign_templates(). See “opcnodegrp_assign_templates()” on page 252 and “opcnodegrp_deassign_nodes()” on page 254 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 templates 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

“OPCDTYPE_TEMPLATE_INFO” on page 486

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	Connect to the OVO 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. 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 250 and “opcnodegrp_deassign_nodes()” on page 254 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_NODE_GROUP” on page 483

Node Hierarchy Configuration API

The Node Hierarchy Configuration API provides a set of functions to configure OVO node hierarchies. To use these functions, it is necessary to connect to the management server as administrator, (see *opc_connect(3)*).

A node hierarchy is specified either by name or by the uuid. If the uuid is given, a specified name will be ignored.

Error information is written to the error logfile `/var/opt/OV/log/OpC/mgmt_sv/opcerror` 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.

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect().

Data Structures

`OPCDTYPE_NODEHIER`

`OPCDTYPE_LAYOUT_GROUP`

`OPCDTYPE_NODE`

Usage

The Node Hierarchy Configuration API can be called by any user.

Prerequisites

The Node Hierarchy Configuration API is only available on the management server.

Multithread Usage

All functions of the OVO Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads, DCE User 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 OVO database
nodehier	Node hierachy 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 ID 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_DATABASE_ERROR	Access to database failed.
OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Hierarchy name already exists.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

“opcdata_create()” on page 52

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 OVO database
nodehier	Node hierachy 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 node hierarchy. The parent layout group defines the location for this layout group. If the parent layout group is NULL, the layout group will be created in node hierarchy toplevel. 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. The name of the layout group must be specified. The node hierarchy must be specified either by name or by uuid.

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 ID 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

“opcdata_create()” on page 52

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 OVO database
src_nodehier	Node hierachy configuration of type OPCDTYPE_NODEHIER.
dst_nodehier	Node hierachy 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

opcnodehier_delete()

```
#include opcsvapi.h

int opcnodehier_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        nodehier  /* in/out */
      );
```

Parameters

opc_conn	Connection to the OVO database
nodehier	Node hierachy 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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 name of the layout group must be given.

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_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 HoldingArea.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy configuration of type OPCDTYPE_NODEHIER.
nodehier_conf	Node hierachy 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy configuration of type OPCDTYPE_NODEHIER.
layoutgrp_list	Container of type OPCDTYPE_EMPTY or OPCDTYPE_LAYOUT_GROUP. The returned elements or 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.

The layout group must exist and be specified by either uuid or name. 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 / wrong type.
OPC_ERR_INVALID_OUTPARAM	node_list is NULL / wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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. 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 / wrong type.
OPC_ERR_INVALID_OUTPARAM	node_list is NULL / wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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 name of the layout group must be given.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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 must exist and be specified by either uuid or name.

If the parent layout group is empty, the top level layout groups of this node hierarchy will be returned. 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 / wrong type.
OPC_ERR_INVALID_OUTPARAM	layout_list is NULL / wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.

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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy configuration of type OPCDTYPE_NODEHIER.
node	Node configuration of type OPCDTYPE_NODE.
parent_group	Layout group configuration of type OPCDTYPE_LAYOUT_GROUP

Description

Moves each node in the list to the specified layout group. 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.

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 / wrong type.
OPC_ERR_INVALID_OUTPARAM	node_list is NULL / wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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 be specified by either the uuid or name. The uuid will supersede the name, if both are given.

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 / wrong type.
OPC_ERR_INVALID_OUTPARAM	node_list is NULL / wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy configuration of type OPCDTYPE_NODEHIER.
nodehier_conf	Node hierachy 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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 name of the layout group must be given.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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. The uuid will supersede the name, if both are given.

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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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. The uuid will supersede the name, if both are given.

The layout groups 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_INVALID_OUTPARAM	layoutgrp_list is NULL / wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

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 OVO database
nodehier	Node hierachy 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.

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. The uuid will supersede the name, if both are given.

Return Values

OPC_ERR_OK	Access successful.
OPC_ERR_INVALID_INPARAM	opc_conn, nodehier, layoutgrp are NULL / wrong type.
OPC_ERR_INVALID_OUTPARAM	node_list is NULL / wrong type.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
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

ITO A.05.00 and later

See Also

“OPCDTYPE_LAYOUT_GROUP” on page 459

“OPCDTYPE_NODE” on page 472

“OPCDTYPE_NODEHIER” on page 484

“opc_connect()” on page 168

Template Configuration API

The Template Configuration API provides a set of functions to configure message source templates and template groups.

General information about templates, for example, name, description, type, and ID, is contained in opcdata structures of the type OPCDTYPE_TEMPLATE_INFO, and can be obtained with the function opctempl_get_list(). Because of the complexity of the templates and their conditions, template details are contained in template files, and can be obtained with the function opctemplfile_get().

Note that the concept of holding template details in template files enables you to handle different versions of templates.

The functions to configure templates work with a simple representation of templates contained in opcdata structures of the type OPCDTYPE_TEMPLATE_INFO. You can use this information to specify a template when you want to delete or modify it, or when you need a full description.

A template is identified by its unique ID. In addition, the template name must also be unique for each template type.

Data Structures

OPCDTYPE_TEMPLATE_INFO

Usage

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect, see page 168.

Prerequisites

This API is only available on the management server.

Multithread Usage

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

opctempl_delete()

```
#include opcsvapi.h

int opctempl_delete (
        const opc_connection   opc_conn,          /* in */
        const opcdata          template           /* in */
                    );
```

Parameters

opc_conn	Connect to the OVO database
template	Template specification of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctempl_delete() to remove a template including all conditions from the OVO database.

NOTE Deleting a template also removes all assignments to managed nodes which results in a changed configuration. You must redistribute the templates 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:	template 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_OBJECT_NOT_FOUND:	template is not in the OVO database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctempl_get_list()

```
#include opcsvapi.h

int opctempl_get_list (
       const opc_connection   opc_conn,         /* in */
       opcdata                templates         /* out */
                        );
```

Parameters

opc_conn	Connect to the OVO database
templates	Container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctempl_get_list() to get a list of all configured templates and template groups including name, description, type, and ID. You need this information to manipulate OVO templates stored in template files.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL
OPC_ERR_INVALID_OUTPARAM:	templates 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

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplfile_add()

```
#include opcsvapi.h

int opctemplfile_add (
        const opc_connection   opc_conn,          /* in */
        const char *           file,              /* in */
        opcdata                templates          /* out */
                        );
```

Parameters

opc_conn	Connect to the OVO database
file	Name of the template file
templates	opcdata container with elements of type OPCDTYPE_TEMPLATE_INFO (containing name, description, and ID of the added templates)

Description

Use the function opctemplfile_add( ) to add new templates to the OVO database. The template names must be unique and must not already exist in the database. Already existing templates are ignored and must be modified using the function opctemplfile_modify().

The function also returns a list of template IDs in opcdata structures of the type OPCDTYPE_TEMPLATE_INFO. The template list can then be used to assign the templates to template groups, node groups, or nodes.

NOTE

Each template in the template list has the flag OPCDATA_STATUS set. If you receive the return value OPC_ERR_NOT_COMPLETELY_DONE, check the status flags of the templates to find out which template 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:	templates 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_CANT_OPEN_FILE:	path does not exist
OPC_ERR_NOT_COMPLETELY_DONE:	not all templates could be added; check the status flag of each template. 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:	template description contains syntax errors

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplfile_get()

```
#include opcsvapi.h

int opctemplfile_get (
        const opc_connection   opc_conn,         /* in */
        const opcdata          template,         /* in */
        const char *           file              /* out */
                        );
```

Parameters

opc_conn	Connect to the OVO database
template	Container of type OPCDTYPE_TEMPLATE_INFO
file	Name of the template file

Description

Use the function opctemplfile_get() to get the details of a template configuration. opctemplfile_get() reads the entire template configuration, including the conditions, from the OVO 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 OVO database using the function opctemplfile_modify().

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL template 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_OBJECT_NOT_FOUND:	template is not in the OVO database
OPC_ERR_CANT_WRITE_FILE:	cannot create file

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplfile_modify()

```
#include opcsvapi.h

int opctemplfile_modify (
       const opc_connection   opc_conn,          /* in */
       const char *           file,              /* in */
       opcdata                templates          /* out */
                          );
```

Parameters

opc_conn	Connect to the OVO database
file	Name of the template file
templates	List of the modified templates; container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctemplfile_modify() to modify already existing templates in the OVO database. The assignments of the templates remain the same; only the parameters and conditions of the templates are changed.

NOTE It is not possible to change the name of the template because the template name acts as an identifier for the template.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL, file is NULL or zero length, templates 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_OBJECT_NOT_FOUND:	template does not exist in the OVO database

OPC_ERR_CANT_OPEN_FILE:	template files do not exist
OPC_ERR_NOT_COMPLETELY_DONE:	not all templates could be added; check the status flag of each template
OPC_ERR_SYNTAX_ERROR:	template description contains syntax errors

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplgrp_add()

```
#include opcsvapi.h

int opctemplgrp_add (
         const opc_connection   opc_conn,     /* in */
         opcdata                templgrp      /* in/out */
                     );
```

Parameters

opc_conn	Connect to the OVO database
templgrp	Template group of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctemplgrp_add() to add a new template group to the OVO database. Initially, this new template group does not contain any templates; use the function opctemplgrp_assign_templates() to add templates or template groups to the group.

opctemplgrp_add() creates a new template group with a new ID that is returned in the opcdata templgrp.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL templgrp 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_OBJECT_ALREADY_EXISTS:	template group already exists in the OVO database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplgrp_assign_templates()

```
#include opcsvapi.h

int opctemplgrp_assign_templates (
        const opc_connection       opc_conn,    /* in */
        const opcdata              templgrp,    /* in */
        const opcdata              templates    /* in/out */
                                   );
```

Parameters

opc_conn	Connect to the OVO database
templgrp	Template group of type OPCDTYPE_TEMPLATE_INFO
templates	opcdata container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctemplgrp_assign_templates() to assign templates or template groups to a given template group. The status of the assignment will be returned in the OPCDATA_STATUS field of each element.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL templgrp is NULL or of incorrect type, templates 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_OBJECT_NOT_FOUND:	template group is not in the OVO database
OPC_ERR_NOT_COMPLETELY_DONE:	not all assignments were successful; check the status field of each object in templates, it may contain OPC_ERR_OBJECT_NOT_FOUND

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplgrp_deassign_templates()

```
#include opcsvapi.h

int opctemplgrp_deassign_templates (
       const opc_connection       opc_conn,     /* in */
       const opcdata              templgrp,     /* in */
       const opcdata              templates     /* in */
              );
```

Parameters

opc_conn	Connect to the OVO database
templgrp	Template group of type OPCDTYPE_TEMPLATE_INFO
templates	opcdata container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctemplgrp_deassign_templates() to deassign templates or template groups from a given template group. Note that if a template is not assigned to any other template group, it is moved to the Toplevel template group. The status of the deassignment will be returned in the OPCDATA_STATUS field of each element.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL templgrp is NULL or of incorrect type, templates 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_OBJECT_NOT_FOUND:	template group is not in the OVO database

OPC_ERR_NOT_COMPLETELY_DONE:	not all assignments were successful; check the status field of each object in templates, it may contain OPC_ERR_OBJECT_NOT_FOUND

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplgrp_delete()

```
#include opcsvapi.h

int opctemplgrp_delete (
        const opc_connection       opc_conn,     /* in */
        const opcdata              templgrp      /* in */
              );
```

Parameters

opc_conn	Connect to the OVO database
templgrp	Template group of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctemplgrp_delete() to remove a given template group from the OVO database. Note that this function does *not* remove templates or template groups contained in this group. Templates and template groups that are not contained in any other template group are moved to the Toplevel template group.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL templgrp 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_OBJECT_NOT_FOUND:	template group is not in the OVO database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplgrp_get()

```
#include opcsvapi.h

int opctemplgrp_get (
        const opc_connection   opc_conn,         /* in */
        const opcdata          templgrp,         /* in */
        opcdata                templ_info        /* out */
              );
```

Parameters

opc_conn	Connect to the OVO database
templgrp	Template group of type OPCDTYPE_TEMPLATE_INFO
templ_info	Template group of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctemplgrp_get() to get the configuration of a given template group from the OVO database when you only know the name of the template group. A template group is identified by its name and ID. The ID 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 templates using the function opctemplgrp_get_templates().

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL, templgrp 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_OBJECT_NOT_FOUND:	template group is not in the database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplgrp_get_templates()

```
#include opcsvapi.h

int opctemplgrp_get_templates (
        const opc_connection       opc_conn,     /* in */
        const opcdata              templgrp,     /* in */
        opcdata                    templates     /* out */
              );
```

Parameters

opc_conn	Connect to the OVO database
templgrp	Template group of type OPCDTYPE_TEMPLATE_INFO
templates	opcdata container of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctemplgrp_get_templates() to get a list of templates and template groups assigned to a given template group. 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 templgrp is NULL or of incorrect type
OPC_ERR_INVALID_OUTPARAM:	templates 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_OBJECT_NOT_FOUND:	template group is not in the OVO database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

opctemplgrp_modify()

```
#include opcsvapi.h

int opctemplgrp_modify (
        const opc_connection       opc_conn,     /* in */
        const opcdata              templgrp,     /* in */
        const opcdata              mod_templgrp  /* in */
              );
```

Parameters

opc_conn	Connect to the OVO database
templgrp	Template group to be modified; of type OPCDTYPE_TEMPLATE_INFO
mod_templgrp	New template group definition of type OPCDTYPE_TEMPLATE_INFO

Description

Use the function opctemplgrp_modify() to modify an existing template group in the OVO database. You can only modify the name and description; use opctemplgrp_assign_templates() or opctemplgrp_deassign_templates() to change the list of assigned templates and template groups.

Return Values

OPC_ERR_OK:	OK
OPC_ERR_INVALID_INPARAM:	opc_conn is NULL templgrp 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_OBJECT_NOT_FOUND:	template group is not in the OVO database

Versions

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“OPCDTYPE_TEMPLATE_INFO” on page 486

User Profile Configuration API

The user profile API provides a set of functions to configure OVO user profiles. To use these functions, it is necessary to connect to the management server as administrator, (see *opc_connect(3)*).

A profile is specified either by name or by the uuid. If the uuid is given, a specified name will be ignored.

Error information is written to the error logfile `/var/opt/OV/log/OpC/mgmt_sv/opcerror` 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.

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect().

Data Structures

`OPCDTYPE_USER_CONFIG`

Usage

The User Profile Configuration API can be called by any user.

Prerequisites

The User Profile Configuration API is only available on the management server.

Multithread Usage

All functions of the OVO Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads, DCE User 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 OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Adds a new OVO user (operator or template 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 ID 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_DATABASE_ERROR	Access to database failed.

OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	Profile already exists.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

“opcdata_create()” on page 52

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 OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.
applgrp_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Assigns application groups to a specified OVO profile.

The 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 assign to 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.
appl_list	Container of type OPCDTYPE_APPL_CONFIG.

Description

Assigns applications to a specified OVO profile.

The 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 assign to 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.
profile_list	Container of type OPCDTYPE_USER_CONFIG.

Description

Assigns profiles to a specified OVO profile.

The profile 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 assign to the 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 OVO profile.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“OPCDTYPE_USER_RESP_ENTRY” on page 488

“opc_connect()” on page 168

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 OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.
applgrp_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Deassigns application groups from a specified OVO profile.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.
appl_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Deassigns applications from a specified OVO profile. The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.
profile_list	Container of type OPCDTYPE_USER_CONFIG.

Description

Deassigns profiles from a specified OVO profile.

The profile 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 assign to the 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 OVO profile.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“OPCDTYPE_USER_RESP_ENTRY” on page 488

“opc_connect()” on page 168

opcprofile_delete()

```
#include opcsvapi.h

 opcprofile_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        profile   /* in/out */
      );
```

Parameters

opc_conn	Connection to the OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Deletes the specified profile.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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.

The 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 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 direct assigned applications of the specified `profile`.

The `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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 OVO profiles.

The parameter profile_list must be 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 OVO profile.

The 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 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, or invalid role.
OPC_ERR_INVALID_OUTPARAM	profile_list is NULL or invalid.
OPC_ERR_ACCESS_DENIED	User has no administrator rights.
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

ITO A.05.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 OVO profile.

The 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 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 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“OPCDTYPE_USER_RESP_ENTRY” on page 488

“opc_connect()” on page 168

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 OVO database
profile	User configuration of type OPCDTYPE_USER_CONFIG.
mod_profile	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Modifies the attributes of OVO profiles.

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

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

User Configuration API

The user API provides a set of functions to configure OVO users (an administrator, operators and template administrators). To use these functions, it is necessary to connect to the management server as administrator (see *opc_connect(3)*).

A user is specified either by name or by the uuid. If the uuid is given, a specified name will be ignored.

Error information is written to the error logfile /var/opt/OV/log/OpC/mgmt_sv/opcerror 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.

To use these functions, it is necessary to connect to the management server as administrator using the function opc_connect.

Data Structures

`OPCDTYPE_USER_CONFIG`

Usage

The User Configuration API can be called by any user.

Prerequisites

The User Configuration API is only available on the management server.

Multithread Usage

All functions of the OVO Configuration APIs are safe to be called by multithreaded applications, and are thread-safe for POSIX Threads, DCE User 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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Adds a new OVO user (operator or template 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 ID 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_DATABASE_ERROR	Access to database failed.

OPC_ERR_NO_MEMORY	Memory allocation failed.
OPC_ERR_OBJECT_ALREADY_EXISTS	User name already exists.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

“opcdata_create()” on page 52

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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
applgrp_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Assigns application groups to a specified OVO user.

The 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 assign to the user, 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 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_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 that user.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
appl_list	Container of type OPCDTYPE_APPL_CONFIG.

Description

Assigns applications to a specified OVO user.

The 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 assign to the user, 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 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_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 that user.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

opcuser_assign_nodehier()

```
#include opcsvapi.h

 opcuser_assign_nodehier (
      opc_connection opc_conn, /*        */
      const opcdata  user,     /*        */
      opcdata        nodehier  /*        */
      );
```

Parameters

opc_conn	Connection to the OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
nodehier	Node hierachy configuration of type OPCDTYPE_NODEHIER.

Description

Assigns a node hierarchy to a specified OVO user.

The user must be specified either by name or the UUID.

The node hierarchy 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 484

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
profile_list	Container of type OPCDTYPE_USER_CONFIG.

Description

Assigns profiles to a specified OVO user.

The 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 assign to the 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 OVO user.

The 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 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 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“OPCDTYPE_USER_RESP_ENTRY” on page 488

“opc_connect()” on page 168

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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
applgrp_list	Container of type OPCDTYPE_APPL_GROUP.

Description

Deassigns application groups from a specified OVO user.

The 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 the user, 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 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_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 that user.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
appl_list	Container of type OPCDTYPE_APPL_CONFIG.

Description

Deassigns applications from a specified OVO user.

The 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 the user, 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 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_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 that user.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
profile_list	Container of type OPCDTYPE_USER_RESP_ENTRY.

Description

Deassigns profiles from a specified OVO user.

The 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 assign to the 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_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 for that user.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 OVO user.

The 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 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 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_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 that user.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“OPCDTYPE_USER_RESP_ENTRY” on page 488

“opc_connect()” on page 168

opcuser_delete()

```
#include opcsvapi.h

 opcuser_delete (
      opc_connection opc_conn, /* in/out */
      opcdata        user      /* in/out */
      );
```

Parameters

opc_conn	Connection to the OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Deletes the specified user.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

opcuser_get_applgrps()

```
#include opcsvapi.h

 opcuser_get_applgrps (
      opc_connection opc_conn,     /*        */
      opcdata        user,         /*        */
      opcdata        applgrp_list  /*        */
      );
```

Parameters

opc_conn	Connection to the OVO 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 direct assigned applications of the specified user.

The 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_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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_GROUP” on page 455

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

opcuser_get_appls()

```
#include opcsvapi.h

 opcuser_get_appls (
      opc_connection opc_conn,  /*        */
      opcdata        user,      /*        */
      opcdata        appl_list  /*        */
      );
```

Parameters

opc_conn	Connection to the OVO 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 direct assigned applications of the specified user.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_APPL_CONFIG” on page 452

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO 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 OVO 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
nodehier	Node hierachy configuration of type OPCDTYPE_NODEHIER.

Description

Gets the assigned node hierarchy for a specified OVO user.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_NODEHIER” on page 484

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

opcuser_get_profiles()

```
#include opcsvapi.h

 opcuser_get_profiles (
      opc_connection opc_conn,     /*        */
      opcdata        user,         /*        */
      opcdata        profile_list  /*        */
      );
```

Parameters

opc_conn	Connection to the OVO 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 OVO user.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

opcuser_get_resps()

```
#include opcsvapi.h

 opcuser_get_resps (
      opc_connection opc_conn,  /* in/out */
      opcdata        user,      /*     in */
      opcdata        resp_list  /*    out */
      );
```

Parameters

opc_conn	Connection to the OVO 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 OVO user.

The 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_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

ITO A.05.00 and later

See Also

“OPCDTYPE_CONTAINER” on page 446

“OPCDTYPE_USER_CONFIG” on page 487

“OPCDTYPE_USER_RESP_ENTRY” on page 488

“opc_connect()” on page 168

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 OVO database
user	User configuration of type OPCDTYPE_USER_CONFIG.
mod_user	User configuration of type OPCDTYPE_USER_CONFIG.

Description

Modifies the attributes of OVO users.

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

The 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. The name of the node hierarchy 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.

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_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.

Versions

ITO A.05.00 and later

See Also

“OPCDTYPE_USER_CONFIG” on page 487

“opc_connect()” on page 168

Distribution API

The Distribution API provides a function to distribute OVO agent configuration to managed nodes.

Data Structures

`OPCDTYPE_CONTAINER`

`OPCDTYPE_NODE_CONFIG`

Usage

The Distribution API can be called by any user.

Prerequisites

The Distribution API is only available on the management server.

Multithread Usage

This API is *not* thread-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	Connect to the OVO 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 OVO agent configuration to managed nodes. This API offers the same functionality as when a user distributes templates, actions, monitors, and commands from the Install / Update OVO Software and Configuration window. It cannot be used to install the OVO 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 OVO administrator.
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

OVO A.05.00 and later

Server Synchronization API

The **Server Synchronization API** provides functions that allow you to control access and modification of the OVO configuration data.

Since multiple processes are able to manipulate OVO configuration data, there is a need to control both the configuration data and its modification. In OVO, 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 407 for more detailed information on the way in which OVO synchronizes changes to configuration data.

Data Structures

OPCDTYPE_USER_CONFIG

OPCTYPE_INFORM_USER

Usage

The Server Synchronization API can be called by any user.

Prerequisites

The Server Synchronization API is only available on the management server.

Multithread Usage

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

The Transaction Concept

OVO'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 OVO is also used here, namely; if OVO cannot get a lock, it waits 5 seconds before trying again. It tries to get the lock 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.

OVO 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 OVO'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 OVO is also used here. If a function is unable to get a lock, it waits 5 seconds and tries again. It retries 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 encouters 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 — Connect to the OVO database

user_info — an opcdata object of type OPCDTYPE_INFORM_USER that contains the following fields, which may be modified with opcdata_set_str():

- OPCDATA_TEXT

  the text of the message sent to the user interfaces

- OPCDATA_NAME

  the name of an OVO user: if undefined, all OVO 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 OVO 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

OVO A.05.00 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 OVO library that is used in this version.

Return Values

The returned string has the format:

`HP OpenView <product> <version> (<date>)`

For example:

HP OpenView HP OpenView Operations A.06.00 (03/16/00).

Versions

OVO A.06.00 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 Connect to the OVO database

Description

Synchronizes the server 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 OVO session (within opc_disconnect()). Therefore, it's not really necessary to call it actively.

NOTE opcsync_inform_server() informs only the server processes. It may be useful to keep the OVO server up-to-date each time a change has been performed, but to leave the GUIs uninformed until all changes are done. It may be disturbing for an operator to reload the configuration with each change.

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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“opcsync_inform_user()” on page 413

opcsync_inform_user()

```
#include opcsvapi.h

int opcsync_inform_user (
     const opc_connection opc_conn          /* in */
                    );
```

Parameters

opc_conn Connect to the OVO database

Description

Informs users of any changes since the last execution of this function by displaying a pop-up window in the UI of the affected users.

This routine can be called, if a program that uses the OVO configuration APIs needs to update OVO user interfaces with the modifications it had performed in the database. Each time an object, for example, a node, a message group, a template, and so on, has been added, modified or deleted, the event is logged in an internal list. opcsync_inform_user() flushes this list when it's called and passes the data to interested OVO components.

NOTE opcsync_inform_user() is automatically called at the end of an OVO session (within opc_disconnect()). Therefore, it's not really necessary to call it actively.

NOTE opcsync_inform_user() informs only the OVO Motif-GUIs. It may be useful to keep the OVO server up-to-date each time a change has been performed, see "opcsync_inform_server()" on page 411 for this purpose.

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

OVO A.05.00 and later

See Also

“opc_connect()” on page 168

“opc_disconnect()” on page 170

“opcsync_inform_server()” on page 411

opctransaction_commit ()

```
#include opcsvapi.h

int opctransaction_commit (
          const opc_connection       opc_conn   /* in */
                        )
```

Parameters

opc_conn Connect to the OVO 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

OVO A.05.00 and later

opctransaction_rollback()

```
#include opcsvapi.h

int opctransaction_rollback (
         const opc_connection        opc_conn   /* in */
                     );
```

Parameters

opc_conn Connect to the OVO 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

OVO A.05.00 and later

opctransaction_start ()

```
#include opcsvapi.h

int opctransaction_start (
          const opc_connection        opc_conn   /* in */
                          );
```

Parameters

opc_conn Connect to the OVO 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

OVO A.05.00 and later

4 Examples

In this Chapter

This chapter provides examples explaining how you can use the OVO APIs.

- ❏ Examples of the OVO Interfaces
  - Using the Server MSI API to Connect to the OVO Interface
  - Registering for Message Events from the OVO Interface
  - 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 OVO Interfaces

Using the Server MSI API to Connect to the OVO 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(&msg, OPCDTYPE_EMPTY))
{
    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,
     MSG_TYPE), ”DISK_FULL”))
    {
        t_disk_full = opcdata_get_long(msg,
        OPCDATA_CREATION_TIME)
    }
    else if (!strcmp(opcdata_get_str(msg, MSG_TYPE),
     ”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, MSG_TEXT,
                 "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 OVO Interface

This program registers for all Message Events (MEs) from the OPCSVIF_MSG_EVENTS queue. 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);

  return (ret);
}
```

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 OVO error code to an error text and prints
this *       text to stderr.
*
* Input Parameters:
*       err_code           OVO 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, “OVO 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 OVO 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 OVO
                database” },
            { OPC_ERR_CANT_DISCONNECT, “Cannot disconnect from
                OVO” },
            { 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 OVO 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 OVO DB */
          if((ret=opc_disconnect(&opc_conn)) != OPC_ERR_OK)
               report_error(ret);
}
```

Examples

5 OVO Data Structures

In This Chapter

This chapter lists the OVO 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.

OVO Data Structures

OVO provides a set of data structures which hold information about OVO 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 OVO 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 OVO 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 OVO 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 OVO 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 OVO 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 OVO 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.
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

Table 5-2 OPCDTYPE_ACTION_RESPONSE (Continued)

Attribute	Scope	Type	Properties	Description
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 OVO 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 hierarchy toplevel.
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 OVO 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_INFORM_USER

Table 5-8 lists the attributes that are available for the Inform User data structure.

Table 5-8 OPCDTYPE_INFORM_USER

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the OVO user who will receive the information message.
OPCDATA_TEXT	get/set	str	L	Text of the message sent to the user.

OPCDTYPE_LAYOUT_GROUP

Table 5-9 lists the attributes that are available for the Node Layout Group data structure.

Table 5-9 OPCDTYPE_LAYOUT_GROUP

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the OVO 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 OVO node layout group.
OPCDATA_SYMBOL	get/set	str		Symbol type of the OVO 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 OVO node layout group.

OPCDTYPE_MESSAGE

Table 5-10 lists the attributes that are available for the Message Attribute data structure.

Table 5-10 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_AACTION_ANNOTATE	get/set	long		Defines whether OVO 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 5-10 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 OVO 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

Table 5-10 OPCDTYPE_MESSAGE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_ESCALATED	get	long		Message was escalated: • OPC_ESCALATED_TO • OPC_ESCALATED_FROM The escalation server can be retrieved using OPCDATA_ESCALATION_SERVER.
OPCDATA_NOTIFICATION	get/set	long		Notification.
OPCDATA_TROUBLETICKET	get/set	long		Forward message to Trouble Ticket system.
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 templates, 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 5-10 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_CONSOLE_SRC: MPE/iX console. • OPC_OPCMSG_SRC: opcmsg(1|3) template • 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.

Table 5-10 OPCDTYPE_MESSAGE (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_MSGTEXT	get/set	str	L	Message Text. Default: empty string.
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 OVO 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 OVO manager if this system is part of the OVO Node Bank. Default: local node name.
OPCDATA_OBJECT	get/set	str	L	Object name to use for the OVO 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 OVO 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 OVO message. Default: empty string. Call of the operator-initiated action.

Table 5-10 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 OVO. 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_ESCALATION_SERVER	get	str		The escalation server.
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_ESCALATED_BY	get	str		Name of the operator who escalated the message. Default: empty string.
OPCDATA_NUM_DUPLICATES	get	long		Number of duplicate messages of this message.
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 5-10 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 5-11 lists the attributes that are available for the Message Event data structure.

Table 5-11 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_ESCALATED • OPC_MSG_EVENT_ESCALATED_FROM • 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
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.

Table 5-11 OPCDTYPE_MESSAGE_EVENT (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_SEVERITY	get	long		Contains the current severity of the message.
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.

OPCDTYPE_MESSAGE_GROUP

Table 5-12 lists the attributes that are available for the Message Group data structure.

Table 5-12 OPCDTYPE_MESSAGE_GROUP

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the OVO message group.
OPCDATA_LABEL	get/set	str	L	Specifies the label shown in the GUI.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the OVO message group.
OPCDATA_SYMBOL	get/set	str		Symbol type displayed in the GUI.

OPCDTYPE_MESSAGE_ID

Table 5-13 lists the attributes that are available for the Message ID data structure.

Table 5-13 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 5-14 lists the attributes that are available for the Monitor Message data structure.

Table 5-14 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 5-15 lists the attributes that are available for the Managed Node data structure.

Table 5-15 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 474.
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 5-15 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 5-16 lists the attributes that are available for the Managed Node Configuration data structure.

Table 5-16 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 OVO node.
OPCDATA_LABEL	get/set	str	L	Label of the OVO node.
OPCDATA_IP_ADDRESS	get	long		IP address of the OVO node.
OPCDATA_SYMBOL	get/set	str		Symbol type of the OVO 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_IP_NAME • OPC_NODE_OTHER

Table 5-16 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_MACHINE_TYPE	get/set	long		Type of the system: • OPC_MACHINE_DEC • OPC_MACHINE_DYNIX • OPC_MACHINE_HP10_S700 • OPC_MACHINE_HP10_S800 • OPC_MACHINE_HP11_PA_RISC • OPC_MACHINE_LINUX22 • OPC_MACHINE_LINUX24 • OPC_MACHINE_NETWARE • OPC_MACHINE_NON_IP • OPC_MACHINE_OS400 • OPC_MACHINE_OSF • OPC_MACHINE_OTHER • OPC_MACHINE_RS6000 • OPC_MACHINE_SGI • OPC_MACHINE_SNI • OPC_MACHINE_SUN_SOLARIS • OPC_MACHINE_UNSUPPORTED • OPC_MACHINE_WINNT • OPC_MACHINE_WINNT_INTEL
OPCDATA_CONTROL	get/set	long		Control type of the system: • OPC_NODE_CONTROLLED • OPC_NODE_MONITORED • OPC_NODE_MESSAGE_ALLOWED • OPC_NODE_DISABLED

Table 5-16 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.

Table 5-16 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_CHARSET	get/set	long		Character set of the node: • OPC_CHARSET_ASCII • OPC_CHARSET_ISO88591 • OPC_CHARSET_ROMAN8 • OPC_CHARSET_EBCDIC • OPC_CHARSET_ACP1252 • OPC_CHARSET_OEMCP850 • OPC_CHARSET_OEMCP437 • OPC_CHARSET_NT_ANSI_L1 • OPC_CHARSET_NT_OEM_L1 • OPC_CHARSET_NT_OEM_US • OPC_CHARSET_ISO88592 • OPC_CHARSET_ISO88595 • OPC_CHARSET_ISO88596 • OPC_CHARSET_ISO88597 • OPC_CHARSET_ISO88598 • OPC_CHARSET_ISO88599 • OPC_CHARSET_TIS620 • OPC_CHARSET_SJIS • OPC_CHARSET_EUCJP • OPC_CHARSET_EUC • OPC_CHARSET_UCS2 • OPC_CHARSET_ACP932 • OPC_CHARSET_NT_ANSI_JP • OPC_CHARSET_NT_UNICODE_JP

Table 5-16 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_CHARSET	get/set	long		Character set of the node: • OPC_CHARSET_NT_OEM_JP • OPC_CHARSET_UTF8 • OPC_CHARSET_EUCKR • OPC_CHARSET_EUCTW • OPC_CHARSET_GB2312 • OPC_CHARSET_BIG5 • OPC_CHARSET_ISO885915 • OPC_CHARSET_ROMAN9 • OPC_CHARSET
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.
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_DEPOT_NAME	get/set	str		Path of the depot to install. Only need to set when the installation method is not OPC_INSTALL_OpC.
OPCDATA_DEPOT_NODE_NAME	get/set	str		Node name of the installation server.
OPCDATA_DEPOT_NODE_IP	get/set	long		IP address of the installation server.

Table 5-16 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_DEPOT_NET_TYPE	get/set	long		Type of the network: • OPC_NETWORK_NO_NODE • OPC_NETWORK_IP • OPC_NETWORK_OTHER • OPC_NETWORK_UNKNOWN
OPCDATA_DCE_SECURITY_LEVEL	get/set	long		DCE security level: • OPC_AUTH_NONE • OPC_AUTH_AT_CONNECTION • OPC_AUTH_EACH_CALL • OPC_AUTH_EACH_PACKET • OPC_AUTH_EACH_PACKET_CHECKSUM • OPC_AUTH_EACH_PACKET_ENCRYPTION
OPCDATA_LOG_DIR	get/set	str		Directory for the log files.
OPCDATA_LOG_SIZE	get/set	long		Maximum size in KB of the logfile
OPCDATA_MPE_STREAM_COMMAND	get/set	str		MPE Job Stream facility.
OPCDATA_PORT_RANGE	get/set	str		Port range for use through firewalls.
OPCDATA_SECURITY_TYPE	get/set	long		Type of security used for the node: • OPC_SEC_NONE • OPC_SEC_PUBLIC_KEY • OPC_SEC_SECRET_KEY
OPCDATA_COMPRESSED_PKG_TRANS	get/set	long		Compressed package transfer (TRUE/FALSE).

Table 5-16 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
OPCDATA_DEPOT_ACCESS_METHOD	get/set	long		Installation method of the depot: • OPC_DEPOT_DEFAULT_METHOD • OPC_DEPOT_FTP_METHOD • OPC_DEPOT_SD_METHOD • OPC_DEPOT_RPC_METHOD
OPCDATA_AGENT_VERSION	get	long		Specifies the agent version of the installed agent. Possible values are: • OPC_AGT_VERS_UNKNOWN • OPC_AGT_VERS_OPC_1X • OPC_AGT_VERS_OPC_2X • OPC_AGT_VERS_ITO_30 • OPC_AGT_VERS_ITO_30_LIGHT • OPC_AGT_VERS_ITO_31 • OPC_AGT_VERS_ITO_40
OPCDATA_RPC_VERSION	get	long		Specifies the RPC version of the installed agent. Possible values are: • OPC_AGT_VERS_ITO_3X_DCE • OPC_AGT_VERS_ITO_3X_NCS • OPC_AGT_VERS_ITO_40_DCE • OPC_AGT_VERS_ITO_40_NCS • OPC_AGT_VERS_ITO_40_SUN

Table 5-16 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
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_UNSPEC_COMM • OPC_COMM_RPC_NCS • OPC_COMM_RPC_DCE_TCP • OPC_COMM_RPC_DCE_UDP • OPC_COMM_RPC_SUN_TCP • OPC_COMM_RPC_SUN_UDP • OPC_COMM_SOCKET_TCP • OPC_COMM_SOCKET_UDP • OPC_COMM_OPC_INTERFACE • OPC_COMM_RPC_LOCAL_RPC
OPCDATA_LICENSE_TYPE	get	long		Specifies whether the node needs a license or not. Possible values are: • OPC_LICENSE_TYPE • OPC_NO_LICENSE_TYPE • OPC_HP_LICENSE
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_MLM_NAME	get/set	str		Specifies the name of the mid-level manager.
OPCDATA_BUFLIMIT_ENABLE	get/set	long		Enables (val != 0) or disables (val = 0) the buffer file size limitation on the node.

Table 5-16 OPCDTYPE_NODE_CONFIG (Continued)

Attribute	Scope	Type	Properties	Description
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.
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 5-17 lists the attributes that are available for the Node Group data structure.

Table 5-17 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 OVO node group.
OPCDATA_LABEL	get/set	str	L	Label of the OVO node group; displayed in the GUI.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the OVO 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 5-18 lists the attributes that are available for the Node Hierarchy data structure.

Table 5-18 OPCDTYPE_NODEHIER

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Name of the OVO 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 OVO node hierarchy.
OPCDATA_ID	get	str		UUID of the OVO node hierarchy.
OPCDATA_SYMBOL	get/set	str		Symbol type of the OVO 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_REGROUP_COND

Table 5-19 lists the attributes that are available for the Regroup Condition data structure.

Table 5-19 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 5-20 lists the attributes that are available for the Template data structure.

Table 5-20 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 template or template group.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the template or template group.
OPCDATA_TYPE	get/set	long		Type of the template or template 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 template or template group. This ID is generated when the template is created.

OPCDTYPE_USER_CONFIG

Table 5-21 lists the attributes that are available for the User Configuration data structure.

Table 5-21 OPCDTYPE_USER_CONFIG

Attribute	Scope	Type	Properties	Description
OPCDATA_NAME	get/set	str		Login name of the OVO user.
OPCDATA_LABEL	get/set	str	L	Specifies the user name shown in the GUI.
OPCDATA_PASSWORD	set	str		Password of the OVO user; necessary for login of the OVO user.
OPCDATA_REAL_NAME	get/set	str	L	Real name of the OVO user.
OPCDATA_DESCRIPTION	get/set	str	L	Description of the OVO user.
OPCDATA_SYMBOL	get/set	str		Symbol type displayed in the GUI.
OPCDATA_ID	get	str		UUID of the OVO user. This parameter is set when the user is created.
OPCDATA_USER_ROLE	get/set	long		Role of the OVO 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 OVO user is allowed to perform operator-initiated actions.
OPCDATA_ACKNOWLEDGE	get/set	long		Specifies whether the OVO 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 5-22 lists the attributes that are available for the User Responsibility Entry data structure.

Table 5-22 OPCDTYPE_USER_RESP_ENTR

Attribute	Scope	Type	Properties	Description
OPCDATA_NODEGROUP_ID	get/set	str		Node group ID—this is the identifier of the OVO 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 OVO 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 OVO message group.

opcregcond

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

Table 5-23 lists the registration conditions of the function `opcif_register()`, see also “opcif_register()” on page 111 for more information.

Table 5-23 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

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 OVO 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"?>
```

Instead of UTF-8, the following codesets can be used with Service Navigator:

Table 6-1 Supported Codesets

Language	Codeset
English and Spanish	ISO-8859-1
	ISO-8859-15
	roman8
	UTF-8
	UTF-16-BE
	UTF-16-LE
Japanese	Shift_JIS
	ISO-2022-JP

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 |
                      GetRootCauses |
                      GetImpacts |
                      Registration |
                      AddLoggings |
                      RemoveLoggings |
                      ListLoggings |
                      GetServicesForMessage |
                      GetViewsForOriginalId |
                      Dump
                      )*)>

<!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?)>

<!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 GetRootCauses       (ServiceRef|OperatorRef)>
<!ELEMENT GetImpacts          (ServiceRef, NoCause?)>
<!ELEMENT AddLoggings         (RegCondition+)>
<!ELEMENT RemoveLoggings      (ServiceRef*)>
<!ELEMENT ListLoggings        (ServiceRef*)>
<!ELEMENT Registration        ((RegCondition*|All),
                              WithLabelChanges?,
                              WithAttributeChanges?)>

<!-- 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 All                   EMPTY>
<!ELEMENT Full                  EMPTY>
<!ELEMENT Recursive             EMPTY>
<!ELEMENT WithLabelChanges      EMPTY>
<!ELEMENT WithAttributeChanges  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> • <GetRootCauses> • <GetImpacts> • <Registration> • <AddLoggings> • <RemoveLoggings> • <ListLoggings> • <GetServicesForMessage> • <GetViewsForOriginalId> • <Dump>

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>
<GetRootCauses>	Any number possible.	Gets the root cause for a service problem. It contains the following tags: • <ServiceRef> • <OperatorRef>
<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>

Table 6-3 The Operations Tags (Continued)

Tag	Required?	Description
<Registration>	Any number possible.	Registration. It contains the following tags: • <RegCondition> • <All> • <WithLabelChanges> • <WithAttributeChanges>
<RegCondition>	Any number possible.	Registration condition. It contains the following tags: • <ServiceRef> • <OperatorRef> • <Depth> • <Recursive>
<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 OpenView Service Navigator Value Pack.
<GetServicesForMessage>	Any number possible.	Gets all services that are mapped to a specific message. This only applies to services configured with the HP OpenView Service Navigator Value Pack.
<Dump>	Any number possible.	Dumps the contents of the service engine. This is useful for troubleshooting purposes.
<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 OpenView Service Navigator Value Pack.
<All>	Empty	All allowed objects.

Table 6-3 The Operations Tags (Continued)

Tag	Required?	Description
<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.
<WithLabelChanges>	Empty	For internal use only.
<WithAttributeChanges>	Empty	For internal use only.
<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 498. The `loggings.dtd` is described in "The Loggings Tags" on page 512.

```
<!-- XML DTD for service engine results  -->
<!-- Results is the root element        -->

<!ENTITY % LoggingsDTD SYSTEM "Loggings.dtd">
%LoggingsDTD;

<!ENTITY % OperationsDTD SYSTEM "operations.dtd">
%OperationsDTD;

<!-- Results from the Service Engine -->

<!ELEMENT Results (OK
                  | Error
                  | Assignments
                  | Loggings
                  | StatusChanges
                  | ConfigChanges
                  | LabelChange
                  | AttributeChange
                  | ElementStatus
                  | AssocStatus
                  | Services
                  | MessageServices
                  | OriginalIdViews)*>

<!ELEMENT OK                    EMPTY>
<!ELEMENT Error                 (#PCDATA)>
<!ELEMENT Assignments           (Assignment*)>
<!ELEMENT StatusChanges         ((ElementStatusChange
                                | AssocStatusChange)*)>
<!ELEMENT ConfigChanges         ((Service | Operator)*)>
<!ELEMENT Assignment            (ServiceRef, OperatorRef*)>
<!ELEMENT ElementStatusChange   (ServiceRef | OperatorRef),
                                Status, OldStatus)
```

```
<!ELEMENT AssocStatusChange    ((Dependency | Composition
                               | OperatorAssignment),
                               SourceRef, TargetRef,
                               Status, OldStatus)>
<!ELEMENT ElementStatus        ((ServiceRef | OperatorRef),
                               Status)>
<!ELEMENT AssocStatus          ((Dependency | Composition
                               | OperatorAssignment),
                               SourceRef, TargetRef,
                               Status)>
<!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) >

<!-- EOF -->
```

Table 6-4 The Results Tags

Tag	Required?	Description
<Results>	Required.	Is the root element. It contains the following tags: • <OK> • <Error> • <Assignments> • <Loggings> • <StatusChanges> • <ConfigChanges> • <LabelChange> • <AttributeChange> • <ElementStatus> • <AssocStatus> • <Services> • <MessageServices> • <OriginalIdViews>
<OK>	Any number possible.	OK. Indicates successful operation or result.
<Error>	Any number possible.	Error. Indicates that the operation failed. Contains the error text.
<Assignments>	Any number possible.	Represents the assignments of a service to operator(s). It contains the following tag: • <Assignment>
<StatusChanges>	Any number possible.	Represents the status changes for a service. It contains the following tags: • <ElementStatusChange> • <AssocStatusChange>
<ConfigChanges>	Any number possible.	Represents any configuration changes. It contains the following tags: • <Service> • <Operator>
<LabelChange>	Any number possible.	For internal use only.

Table 6-4 The Results Tags (Continued)

Tag	Required?	Description
<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>
<AssocStatusChange>	Any number possible.	Gets the status change of a service association. It contains the following tags: • <Dependency> • <Composition> • <OperatorAssignment> • <SourceRef> • <TargetRef> • <Status> • <OldStatus>
<ElementStatus>	Any number possible.	Gets the status change of a service element. It contains the following tags: • <ServiceRef> • <OperatorRef> • <Status>

Table 6-4 The Results Tags (Continued)

Tag	Required?	Description
<AssocStatus>	Any number possible.	Gets the status change of a service element. It contains the following tags: • <Dependency> • <Composition> • <OperatorAssignment> • <SourceRef> • <TargetRef> • <Status>
<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 OpenView Service Navigator Value Pack.
<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 OpenView Service Navigator Value Pack.
<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.

Table 6-4 The Results Tags (Continued)

Tag	Required?	Description
<TopLevelService>	Required.	Top-level service in a service hierarchy that has a subservice with the specified original ID. It contains the following tag: • <Service>
<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 498.

```
<!-- 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 513 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.
<AssocStatus>	Status of an association.
<Check>	OK.
<DeassignServices>	OK.
<Dump>	OK.
<ElementStatus>	Status of an element.
<GetImpacts>	Services.
<GetRootCauses>	Services.
<GetServicesForMessage>	Services.
<GetViewsForOriginalId>	Service hierarchies with topmost service and its subservice that has the specified original ID.
<List>	Services.
<ListAssignments>	Assignments.
<ListLoggings>	Loggings.
<Remove>	OK.
<RemoveAttributes>	OK.
<RemoveLoggings>	OK.

Table 6-6 Results (Continued)

Operation	Result
<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();
  Registration *NewRegistration();
  Service      *NewService( char const *svc );
  void Register( Registration *reg );
  void Register();
  void Unregister();
  void Listen( ServiceStatusListener *statusListener = NULL );

  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 APIs for the service registration interface and some APIs for the service operations interface . 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.
`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.

Example

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 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;
};
```

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 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 516.

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`.
`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.
`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
```

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 void AddRegCond( char const *name,int level = 0 )
    = 0;
  virtual void RemoveRegCond( char const *name = NULL ) = 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 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 method ServiceStatusListener::ServiceStatusChanged(). 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.

The ServiceStatusListener Class

Prototype

```
class ServiceStatusListener
{
 public:
  virtual void ServiceStatusChanged
    ( vector<ServiceStatusChange *> 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. 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.

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
```

A About OVO Man Pages

In this Appendix

This appendix describes the man pages available in the following areas:

- Man Pages in OVO
- Man Pages for OVO APIs
- Man Pages for HP OpenView Service Navigator
- Man Pages for the OVO Developer's Kit APIs

Accessing and Printing Man Pages

You can access the OVO man pages from the command line, from online help, or in HTML format on your management server.

To Access an OVO Man Page from the Command Line

To access an OVO man page from the command line, enter the following:

```
man <manpagename>
```

To Print a Man Page from the Command Line

To print an OVO 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 OVO 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 OVO

This section describes man pages in OVO.

Table A-1 OVO Man Pages

Man Page	Description
call_sqlplus.sh(1)	Calls SQL*Plus.
inst.sh(1M)	Installs OVO software on managed nodes.
inst_debug(5)	Debugs an installation of the OVO agent software.
ito_op(1M)	Launches the OVO Java-based operator or Service Navigator GUI.
ito_op_api_cli(1M)	Enables calling the Java GUI Remote APIs.
opc(1|5)	Starts the OVO GUI.
opc_audit_secure(1M)	Locks the audit level in the OVO database, and allows directories for the history and audit download to be set.
opc_backup(1M)	Interactively saves the OVO environment for Oracle.
opc_backup(5)	Backs up the OVO configuration.
opc_chg_ec(1M)	Changes circuit names in event correlation (EC) templates in the OVO database.
opc_recover(1M)	Interactively recovers the OVO environment for Oracle.
opc_recover(5)	Recovers the OVO 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 OVO agent.
opcadddbf(1M)	Adds a new datafile to an Oracle tablespace.

Table A-1 OVO Man Pages (Continued)

Man Page	Description
opcagt(1M)	Administers agent processes on a managed node.
opcagtreg(1M)	Registers subagents.
opcagtutil(1M)	Parses the agent platform file, and performs operations with extracted data.
opcaudupl(1M)	Uploads audit data into the OVO database.
opcaudwn(1M)	Downloads audit data into the OVO database.
opccfgdwn(1M)	Downloads configuration data from the database to flat files.
opccfgout(1M)	Configures condition status variables for scheduled outages in OVO.
opccfgupld(1M)	Uploads configuration data from flat files into the database.
opcchgaddr(1M)	Changes the address of nodes in the OVO database.
opccltconfig(1M)	Configures OVO client filesets.
opcconfig(1M)	Configures an OVO 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 OVO database.
opcdbinit(1M)	Initializes the database with the default configuration.
opcdbinst(1M)	Creates or destroys the OVO database scheme.
opcdbpwd(1M)	Changes the password of the OVO database user opc_op.
opcdbreorg(1M)	Re-organizes the tables in the OVO database.
opcdbsetup(1M)	Creates the tables in the OVO database.

Table A-1 OVO Man Pages (Continued)

Man Page	Description
opcdcode(1M)	Views OVO encrypted template files.
opcerr(1M)	Displays instruction text for OVO 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 OVO history messages to a file.
opchistupl(1M)	Uploads history messages into the OVO database.
opcmack(1)	Acknowledges an OVO message by specifying the message ID.
opcmgrdist(1M)	Distributes the OVO configuration between management servers.
opcmom(4)	Provides an overview of OVO MoM functionality.
opcmomchk(1)	Checks syntax of MoM templates.
opcmon(1)	Forwards the value of a monitored object to the OVO monitoring agent on the local managed node.
opcmsg(1)	Submits a message to OVO.
opcpat(1)	Tests a program for OVO pattern matching.
opcragt(1M)	Remotely administers agent services for OVO on a managed node.
opcskm(3)	Manages secret keys.
opcsqlnetconf(1M)	Configures the OVO database to use an Net8 connection.
opcsv(1M)	Administers OVO manager services.
opcsvreg(1M)	Registers server configuration files.
opcsvskm(1M)	Manages secret keys on the management server.
opcsw(1M)	Sets the software status flag in the OVO database.
opcswitchuser(1M)	Switches the ownership of the OVO agents.

Table A-1 OVO Man Pages (Continued)

Man Page	Description
opctempl(1M)	Maintains templates in files.
opctemplate(1M)	Enables and disables templates.
opctmpldwn(1M)	Downloads and encrypts OVO message source templates.
opcwall(1)	Sends a message to currently logged in OVO users.
ovocomposer(1M)	Performs tasks related to OV Composer.
ovocomposer(5)	Describes the Correlation Composer, an HP OpenView Operations (OVO) event correlation feature.
ovtrap2opc(1M)	Converts the `trapd.conf` file and the OVO template file.

Man Pages for OVO APIs

This section describes man pages for OVO application program interfaces (APIs).

Table A-2 OVO API Man Pages

Man Page	Description
opcmon(3)	Forwards the value of a monitored object to the OVO monitoring agent on the local managed node.
opcmsg(3)	Submits a message to OVO.

Man Pages for HP OpenView Service Navigator

This section describes man pages for the HP OpenView Service Navigator.

Table A-3 Service Navigator Man Pages

Man Page	Description
`opcservice(1M)`	Configures HP OpenView Service Navigator.
`opcsvcattr (1M)`	Add, change or remove service attributes.
`opcsvcconv(1M)`	Converts service configuration files of HP OpenView Service Navigator from the previous syntax to the Extensible Markup Language (XML).
`opcsvcdwn(1M)`	Downloads service status logs of HP OpenView Service Navigator to a file.
`opcsvcterm(1M)`	Emulates an interface to HP OpenView 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 OpenView Service Navigator into the OVO database.

Man Pages for the OVO Developer's Kit APIs

This section describes man pages for the OVO Developer's Kit application program interfaces (APIs).

Table A-4 OVO Developer's Toolkit Man Pages

Man Page	Description
msiconf(4)	Configures the OVO message manager.
opc_comif_close(3)	Closes an instance of the communication queue interface.
opc_comif_freedata(3)	Displays free data that was allocated by opc_comif_read().
opc_comif_open(3)	Opens an instance of the communication queue interface.
opc_comif_read(3)	Reads information from a queue.
opc_comif_read_request(3)	Reads information from a queue.
opc_comif_write(3)	Writes information into a queue.
opc_comif_write_request(3)	Writes information into a queue.
opc_connect_api(3)	Connects OVO.
opc_distrib(3)	Distributes the OVO agent configuration.
opcagtmon_send(3)	Forwards the value of a monitored object to OVO.
opcagtmsg_api(3)	Handles messages on OVO agents.
opcanno_api(3)	Manages OVO message annotations.
opcapp_start(3)	Starts an OVO application.
opcappl_api(3)	Configures and starts OVO applications.
opcapplgrp_api(3)	Configures OVO application groups.
opcconf_api(3)	Gets OVO configuration.

Table A-4 OVO Developer's Toolkit Man Pages (Continued)

Man Page	Description
opcdata(3)	Accesses the attributes of the OVO data structure.
opcdata_api(3)	Describes how to access the OVO data structure using the OVO Data API.
opcif_api(3)	API to work with the OVO Message Stream Interface.
opciter(3)	OVO iterator to step through opcdata container.
opcmsg_api(3)	Manages OVO messages.
opcmsggrp_api(3)	Manages OVO message groups.
opcmsgregrpcond_api(3)	Creates and modifies OVO message regroup conditions.
opcnode_api(3)	Configures OVO managed nodes.
opcnodegrp_api(3)	Configures OVO node groups.
opcnodehier_api(3)	Configures OVO node hierarchies.
opcprofile_api(3)	Configures OVO user profiles.
opcregcond(3)	Accesses fields of the OVO registration condition structure.
opcsvc_api(3)	C++ classes for Service Navigator.
opctempl_api(3)	Configures OVO message source templates.
opctempfile_api(3)	Configures OVO templates using template files.
opctemplgrp_api(3)	Configures OVO template groups.
opctransaction_api(3)	Starts, commits, and rolls back transactions.
opcuser_api(3)	Configures OVO users.
opcversion(3)	Returns the string of the OVO library that is currently used.

B API Changes

In this Appendix

This appendix lists the API changes in the OVO Developer's Toolkit:

- Changes from OVO A.06.xx to A.07.00
- Changes from OVO A.07.xx to A.08.10

Changes from OVO A.06.xx to A.07.00

This section lists the changes in the OVO Developer's Toolkit between versions A.06.xx and A.07.00:

API Changes between A.06.xx and A.07.00

- New APIs
- Changed OVO Data Structures

Table B-1 New APIs

API	Description
`opcdata_get_cma()`	The function `opcdata_get_cma()` is new with A.07.00. See "opcdata_get_cma()" on page 56 for more information.
`opcdata_get_cmanames()`	The function `opcdata_get_cmanames()` is new with A.07.00. See "opcdata_get_cmanames()" on page 58 for more information.
`opcdata_remove_cma()`	The function `opcdata_remove_cma()` is new with A.07.00. See "opcdata_remove_cma()" on page 74 for more information.
`opcdata_set_cma()`	The function `opcdata_set_cma()` is new with A.07.00. See "opcdata_set_cma()" on page 76 for more information.

Table B-2 Changed OVO Data Structures

OVO Data Structure	Description

OPCDTYPE_NODE_CONFIG	The data structure OPCDTYPE_NODE_CONFIG, see Table 5-16, "OPCDTYPE_NODE_CONFIG," on page 474 was changed as follows:
	• The following values of the attribute OPCDATA_MACHINE_TYPE are obsolete:
	  • OPC_MACHINE_LINUX
	  • OPC_MACHINE_NCR
	  OPC_MACHINE_UNIXWARE
	  OPC_MACHINE_WINNT_ALPHA
	• The following values of the attribute OPCDATA_MACHINE_TYPE are new:
	  • OPC_MACHINE_LINUX24
	• The following values of the attribute OPCDATA_CHARSET are new:
	  • OPC_CHARSET_ISO885915
	  • OPC_CHARSET_ROMAN9

Changes from OVO A.07.xx to A.08.10

This section lists the changes in the OVO Developer's Toolkit between versions A.07.xx and A.08.10:

API Changes between A.07.xx and A.08.10

The following APIs are introduced with the version A.07.10.1:

- `SetAttributes()`
- `RemoveAttributes()`
- `RemovePrefAttributes()`

See "The Service Class" on page 519 and "The Operations Tags" on page 498 for more information about these new functions.

T

U

X

XML notation, 494