Dialogic
Making Innovation Thrive

# Dialogic® NaturalAccess™ OAM API Developer's Manual

October 2009 64-0492-01

www.dialogic.com

## Copyright and legal notices



All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

Dialogic, Dialogic Pro, Brooktrout, Diva, Cantata, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, Diva ISDN, TruFax, Exnet, EXS, SwitchKit, N20, Making Innovation Thrive, Connecting to Growth, Video is the New Voice, Fusion, Vision, PacketMedia, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation or its subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. The names of actual companies and product mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

## Revision history

| Revision | Release date | Notes |
|---|---|---|
| 9000-6818-10 | February, 2000 | CYF, for CT Access 4.0 Beta |
| 9000-6818-11 | July, 2000 | CYF, for PSF 4.0 release |
| 9000-6818-12 | September, 2000 | CYF, for CT Access 4.0 |
| 9000-6818-13 | February, 2001 | CYF, for Natural Access 2000-2 |
| 9000-6818-14 | April, 2001 | CYF, for Natural Access 2001-1 Beta |
| 9000-6818-15 | June, 2001 | CYF |
| 9000-6818-16 | August, 2001 | CYF, for Natural Access 2001-1 |
| 9000-6818-17 | November, 2001 | CYF, for Natural Access 2002-1 Beta |
| 9000-6818-18 | May, 2002 | MVH, for Natural Access 2002-1 |
| 9000-6818-19 | April, 2003 | SRR, for Natural Access 2003-1 |
| 9000-6818-20 | November, 2003 | MVH, for Natural Access 2004-1 Beta |
| 9000-6818-21 | April, 2004 | MCM, for Natural Access 2004-1 |
| 9000-6818-22 | June, 2008 | LBZ, Natural Access R8 |
| 9000-6818-23 | February, 2009 | DEH, Natural Access R8.1 |
| 64-0492-01 | October 2009 | LBG, NaturalAccess R9.0 |
| Last modified: September 2, 2009 | | |

Refer to www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

# Table Of Contents

# 1 Introduction

The *Dialogic® NaturalAccess™ OAM API Developer's Manual* provides:

- An overview of the NaturalAccess OAM API
- A reference of functions, errors, events, and reasons

This manual defines terms where applicable, but assumes that you are familiar with the C programming language.

# 2 Terminology

**Note:** The product to which this document pertains is part of the NMS Communications Platforms business that was sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") on December 8, 2008. Accordingly, certain terminology relating to the product has been changed. Below is a table indicating both terminology that was formerly associated with the product, as well as the new terminology by which the product is now known. This document is being published during a transition period; therefore, it may be that some of the former terminology will appear within the document, in which case the former terminology should be equated to the new terminology, and vice versa.

| Former terminology | Dialogic terminology |
|---|---|
| CG 6060 Board | Dialogic® CG 6060 PCI Media Board |
| CG 6060C Board | Dialogic® CG 6060C CompactPCI Media Board |
| CG 6565 Board | Dialogic® CG 6565 PCI Media Board |
| CG 6565C Board | Dialogic® CG 6565C CompactPCI Media Board |
| CG 6565e Board | Dialogic® CG 6565E PCI Express Media Board |
| CX 2000 Board | Dialogic® CX 2000 PCI Station Interface Board |
| CX 2000C Board | Dialogic® CX 2000C CompactPCI Station Interface Board |
| AG 2000 Board | Dialogic® AG 2000 PCI Media Board |
| AG 2000C Board | Dialogic® AG 2000C CompactPCI Media Board |
| AG 2000-BRI Board | Dialogic® AG 2000-BRI Media Board |
| NMS OAM Service | Dialogic® NaturalAccess™ OAM API |
| NMS OAM System | Dialogic® NaturalAccess™ OAM System |
| NMS SNMP | Dialogic® NaturalAccess™ SNMP API |
| Natural Access | Dialogic® NaturalAccess™ Software |
| Natural Access Service | Dialogic® NaturalAccess™ Service |
| Fusion | Dialogic® NaturalAccess™ Fusion™ VoIP API |
| ADI Service | Dialogic® NaturalAccess™ Alliance Device Interface API |
| CDI Service | Dialogic® NaturalAccess™ CX Device Interface API |
| Digital Trunk Monitor Service | Dialogic® NaturalAccess™ Digital Trunk Monitoring API |
| MSPP Service | Dialogic® NaturalAccess™ Media Stream Protocol Processing API |
| Natural Call Control Service | Dialogic® NaturalAccess™ NaturalCallControl™ API |
| NMS GR303 and V5 Libraries | Dialogic® NaturalAccess™ GR303 and V5 Libraries |

| Former terminology | Dialogic terminology |
|---|---|
| Point-to-Point Switching Service | Dialogic® NaturalAccess™ Point-to-Point Switching API |
| Switching Service | Dialogic® NaturalAccess™ Switching Interface API |
| Voice Message Service | Dialogic® NaturalAccess™ Voice Control Element API |
| NMS CAS for Natural Call Control | Dialogic® NaturalAccess™ CAS API |
| NMS ISDN | Dialogic® NaturalAccess™ ISDN API |
| NMS ISDN for Natural Call Control | Dialogic® NaturalAccess™ ISDN API |
| NMS ISDN Messaging API | Dialogic® NaturalAccess™ ISDN Messaging API |
| NMS ISDN Supplementary Services | Dialogic® NaturalAccess™ ISDN API Supplementary Services |
| NMS ISDN Management API | Dialogic® NaturalAccess™ ISDN Management API |
| NaturalConference Service | Dialogic® NaturalAccess™ NaturalConference™ API |
| NaturalFax | Dialogic® NaturalAccess™ NaturalFax™ API |
| SAI Service | Dialogic® NaturalAccess™ Universal Speech Access API |
| NMS SIP for Natural Call Control | Dialogic® NaturalAccess™ SIP API |
| NMS RJ-45 interface | Dialogic® MD1 RJ-45 interface |
| NMS RJ-21 interface | Dialogic® MD1 RJ-21 interface |
| NMS Mini RJ-21 interface | Dialogic® MD1 Mini RJ-21 interface |
| NMS Mini RJ-21 to NMS RJ-21 cable | Dialogic® MD1 Mini RJ-21 to MD1 RJ-21 cable |
| NMS RJ-45 to two 75 ohm BNC splitter cable | Dialogic® MD1 RJ-45 to two 75 ohm BNC splitter cable |
| NMS signal entry panel | Dialogic® Signal Entry Panel |

# 3 Overview of the NMS OAM service

## NMS OAM overview

NMS OAM can manage hardware components, such as NMS boards, or software components, such as the NMS Hot Swap process. Resources can be located on one CPU, or on multiple CPUs connected over an IP network. Components being managed by NMS OAM are called managed components.

You can access NMS OAM functionality in one of the following ways:

- Using the NMS OAM service. The NMS OAM service is a standard Natural Access service. This manual describes how to access NMS OAM this way.
- Using the *oamsys*, *oamcfg*, *oammon*, and *oaminfo* utilities included with NMS OAM. For more information on these utilities, refer to the *NMS OAM System User's Manual*.

You can perform the following tasks using NMS OAM:

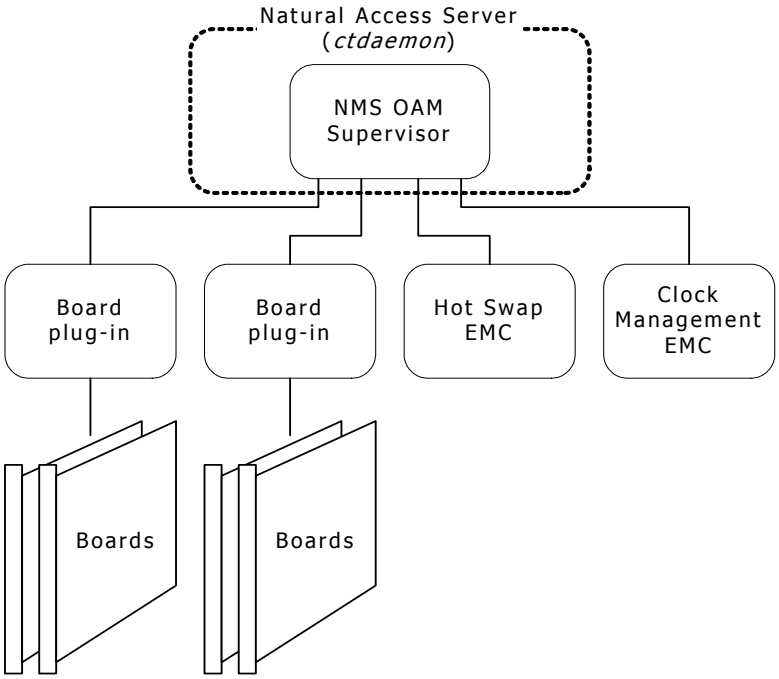
- Create, delete, and query component configurations
- Start, stop, and test components
- Receive notifications from components
- Detect boards in a system
- Perform detailed management of configuration settings
- Export or import configuration data

## NMS OAM components

The NMS OAM software includes the following components:

- NMS OAM Supervisor
- Board plug-ins
- Extended management components (EMCs)

The following illustration shows these components:




### NMS OAM Supervisor

The Supervisor provides the main NMS OAM logic. It performs the following tasks:

- Loads all board plug-ins and extended management components (EMCs) when it starts up.
- Coordinates the activities of the managed components.
- Manages a database containing configuration information for the managed components (described in *Configuration database* on page 14).

The NMS OAM Supervisor is an integral part of the Natural Access Server (*ctdaemon*). To use NMS OAM, Natural Access must be installed on your system and *ctdaemon* must be running. To learn how to start *ctdaemon*, refer to the *NMS OAM System User's Manual*.

### Board plug-ins

NMS OAM communicates with boards through software extensions called board plug-ins, one for each board family. The NMS OAM board plug-ins support the AG, CG, CX, and QX PCI and CompactPCI board models. The board plug-ins do not support TX boards.

When the Supervisor starts up, it loads all plug-ins that it finds on the system. The Supervisor looks for these modules in the *nms\bin* directory (*/opt/nms/lib* under UNIX). Plug-in files have the *.bpi* extension.

### Extended management components (EMCs)

EMCs are software modules that add functionality to NMS OAM. NMS OAM provides the following EMCs:

- The Hot Swap EMC enables you to insert and extract Hot Swap-compatible CompactPCI boards without powering down the system. Hot Swap capability improves system availability by reducing downtime due to routine configuration changes and board replacements.
- The Clock Management EMC configures and manages H.100 and H.110 bus clocking.

When the Supervisor starts up, it loads all EMCs that it finds on the system. The Supervisor looks for these modules in the *\nms\bin* directory (*/opt/nms/lib* under UNIX). EMC files have the *.emc* extension.

## Managed objects

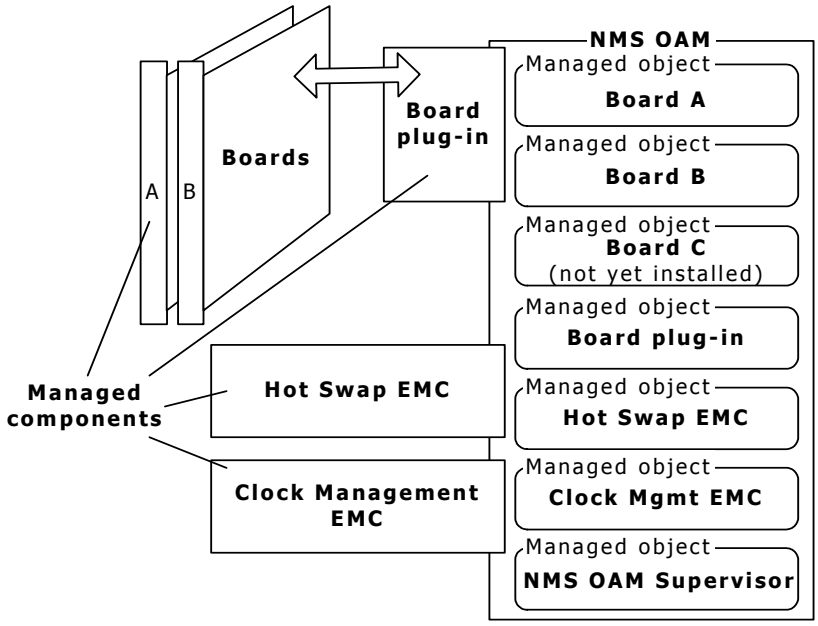
All components (board plug-ins and EMCs) logically exist as managed objects within NMS OAM. NMS OAM records parameters for each object in a configuration database. A managed object is the logical representation of a managed component to NMS OAM.

Boards are also logically represented as managed objects. A board must exist as a managed object in order for NMS OAM to configure or start it.

NMS OAM can have managed objects that do not correspond to active or present managed components in the system. For example, you can have a managed object for an NMS board that is not in the chassis, although an error results if an attempt is made to start that component. Conversely, not all NMS resources in the system necessarily exist as managed objects within NMS OAM.

The NMS OAM Supervisor has a managed object. You can access the Supervisor managed object to query and configure various system-level parameters. For more information, refer to *Supervisor keyword summary* on page 117.

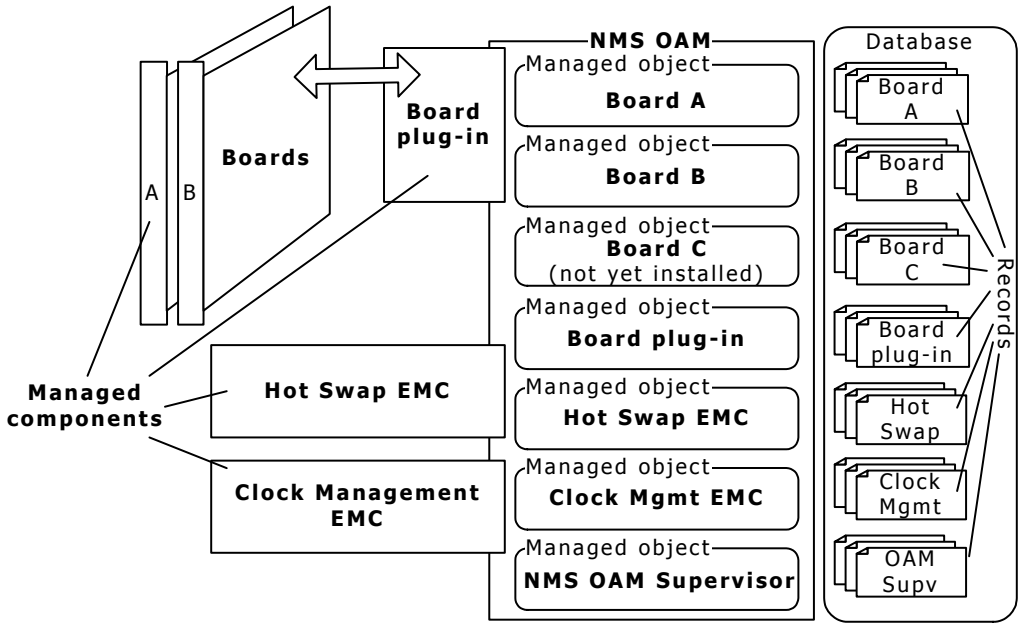
The following illustration shows the managed objects and components:




## Configuration database

NMS OAM maintains a configuration database to facilitate the management of the components under its control. Each hardware and software managed object has its own record in the database containing configuration parameters and parameter values.

The following illustration shows the OAM configuration database:




In the database, each parameter and value is expressed as a keyword name/value pair (for example, AutoStart = YES). You can use the NMS OAM service to query or modify keyword values for any managed object.

## NMS OAM and Natural Access

NMS OAM is an integral part of Natural Access. Natural Access is a development environment providing standard programming interfaces for hardware-independent functions. You must have Natural Access installed on each host system to build applications using the NMS OAM service.

For detailed information about Natural Access, see the *Natural Access Developer's Reference Manual*.

### Client/server Natural Access

Natural Access is based on a client/server model. This model allows client applications and servers to share resources and distribute processing tasks over a network. Clients and servers can reside on the same chassis or on separate chassis. Client applications can access resources available on multiple Natural Access chassis.

NMS OAM operates within this model. Regardless of whether an application is accessing NMS OAM on a local or remote system, the relationship between the application and NMS OAM is always client and server.

NMS OAM is built into the Natural Access Server (*ctdaemon*). Thus, *ctdaemon* must be running on a system in order for NMS OAM to be operational.

### Natural Access services

NMS OAM is implemented as a Natural Access service. A service is a group of logically related functions. The NMS OAM service provides functions to manage and maintain NMS resources. Other services address other aspects of an application. For example, the Switching service controls circuit switching; the NaturalFax service provides fax functionality.

### Natural Access contexts

A context organizes services and accompanying resources around a single processing unit. To access NMS OAM service functionality on a given host system, a client application creates a context on that system and opens the NMS OAM service on the context, along with any other services it requires. For more information, refer to *Initializing NMS OAM client applications* on page 31.

**Note:** A context can access the services and resources only on the host system on which it was created.

### Event queues

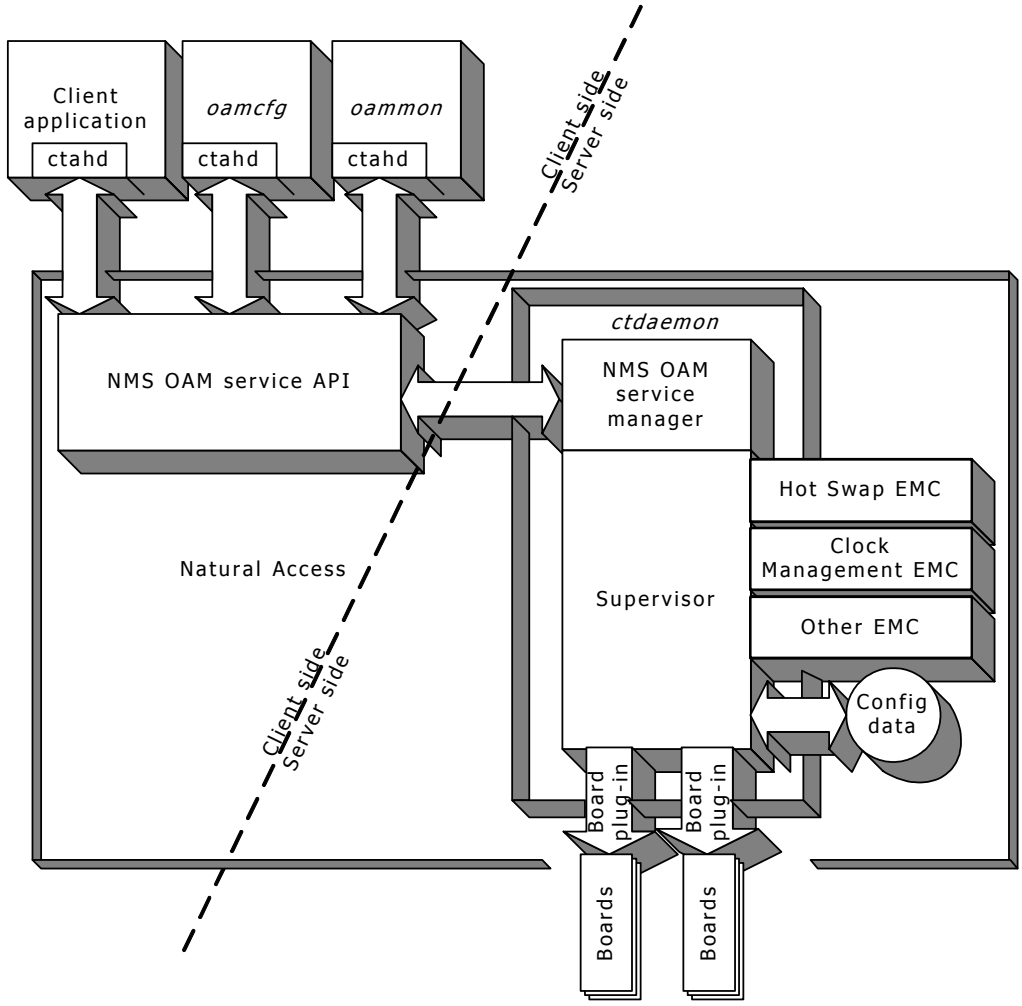
An event queue is the communication path from a service to an application. A service generates events indicating certain conditions or state changes. An application retrieves the events from the event queue.

When creating a context, you specify an event queue. All events from the services on the context are sent to the event queue attached to the context.

You can specify the same event queue for more than one context, even if the contexts were created on different host systems.

## NMS OAM service architecture

The following illustration shows the architecture of the NMS OAM service:




The client side and server side can reside in the same system, or in different systems linked by an IP network. The client side contains one or more client applications that communicate with NMS OAM through its API, using one or more contexts. An NMS OAM service instance is started on each context. Each instance communicates with the NMS OAM service manager component of the Natural Access Server (*ctdaemon*) on the server side.

The *oamcfg*, *oammon*, and *oaminfo* utilities are client applications. *oamsys* is not a client application, but it uses multiple calls to *oamcfg* to perform its functions. For information about these utilities, refer to the *NMS OAM System User's Manual*.

The server side contains the Natural Access Server (*ctdaemon*), which has the following components:

| Component | Description |
|---|---|
| NMS OAM service manager | Provides the glue logic between NMS OAM service instances and the Supervisor. |
| NMS OAM Supervisor | Manages the configuration database, board plug-ins and EMCs. |

## Multiple-host NMS OAM

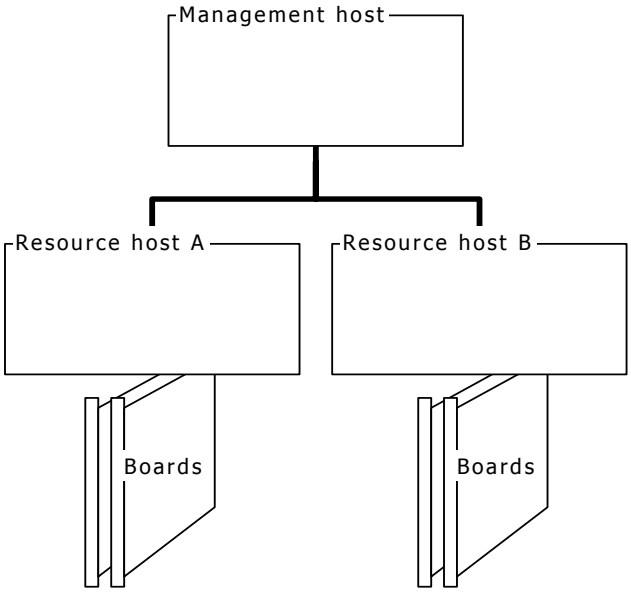
This topic describes the following elements of NMS OAM multiple-host configuration:

- Management host and resource hosts
- Host configuration
- Configuring the NMS OAM database
- Multiple-server configuration

### Management host and resource hosts

If your resources are distributed over several systems (multiple CPUs, chassis, or both) that are linked over an IP network, you can set up a multiple-host configuration of NMS OAM. This configuration enables an application running on one system to access and manage resources on other systems.

A multiple-host NMS OAM configuration consists of several hosts (CPUs). The management host configures and manages the resources (such as boards) on the other hosts. These hosts are called resource hosts as shown in the following illustration:

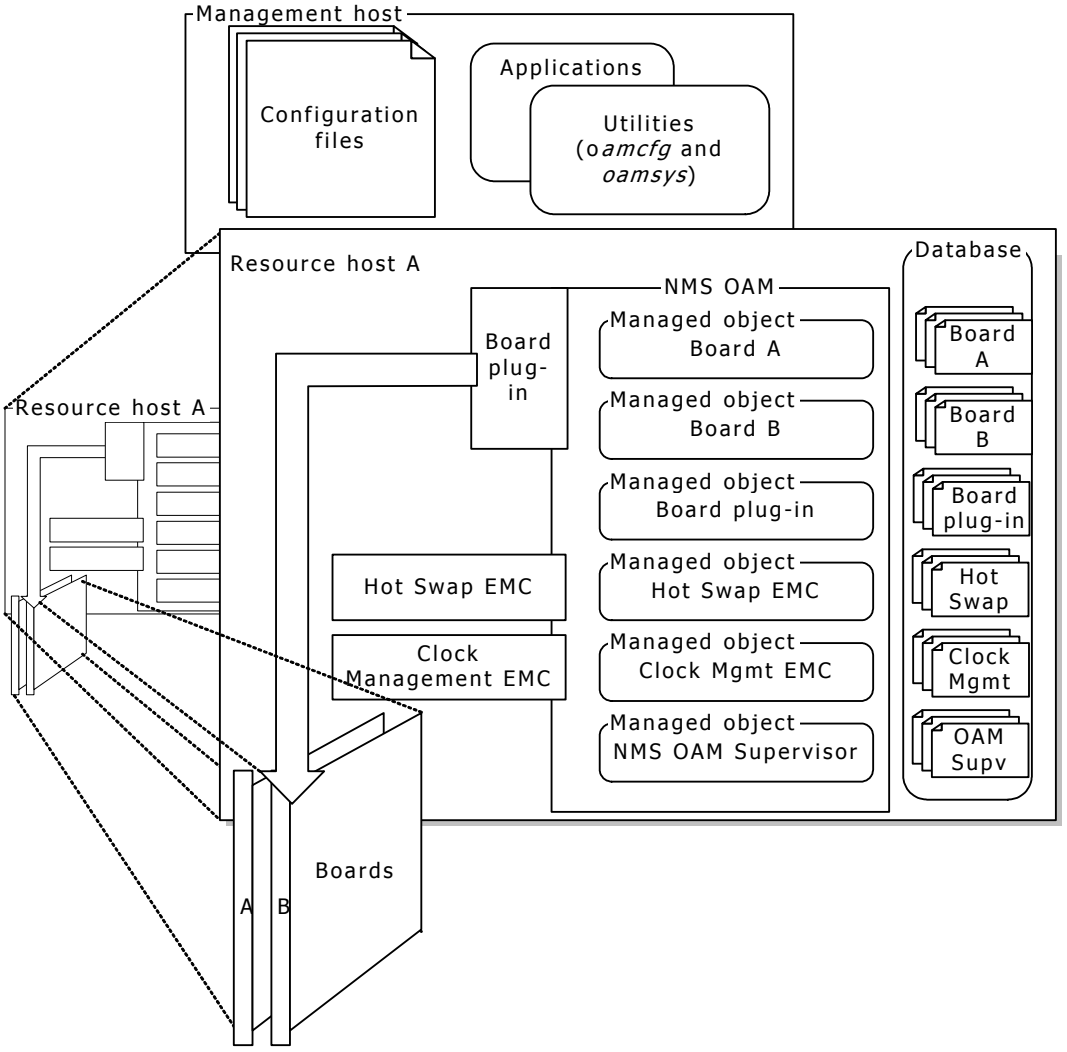



### Host configuration

Each resource host runs an instance of the Natural Access Server (*ctdaemon*), including NMS OAM. Each resource host has its own set of managed objects for its components, including boards, plug-ins, and EMCs. Each resource host also has its own NMS OAM database containing data for the components on that host only.

The Supervisor component on a resource host manages only the resources locally present on the host.

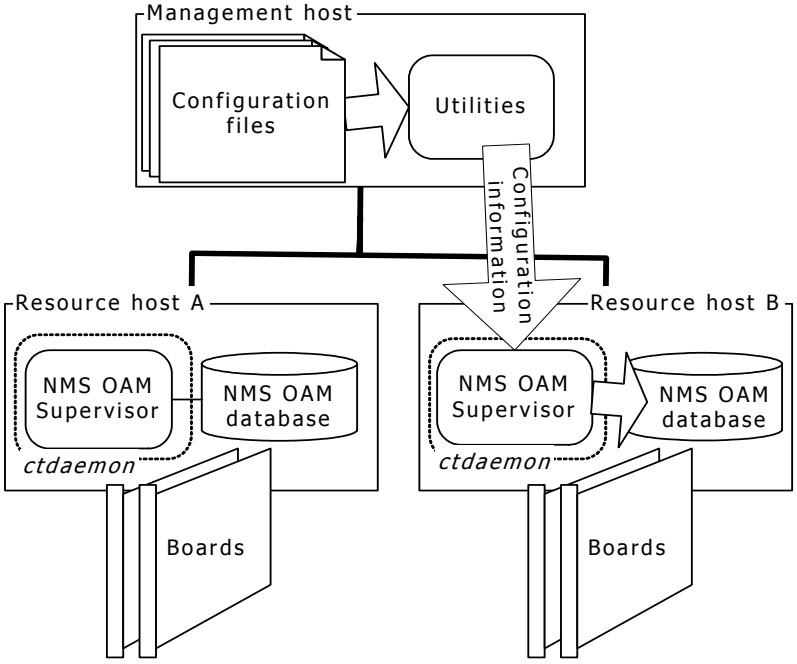
Management applications and utilities such as the NMS OAM utilities reside on the management host. The configuration text files also reside on the management host:




If a management host includes resources that require management, it can also serve as a resource host. When acting as a resource host, the management host must also run an instance of the Natural Access Server (*ctdaemon*), including NMS OAM. The database on this host contains information about the local resources only, even if the management host is also managing other resource hosts.

## Configuring the NMS OAM database

Configuration of remote resources is performed from the management host using NMS utilities or other applications based upon the NMS OAM service, as shown in the following illustration:

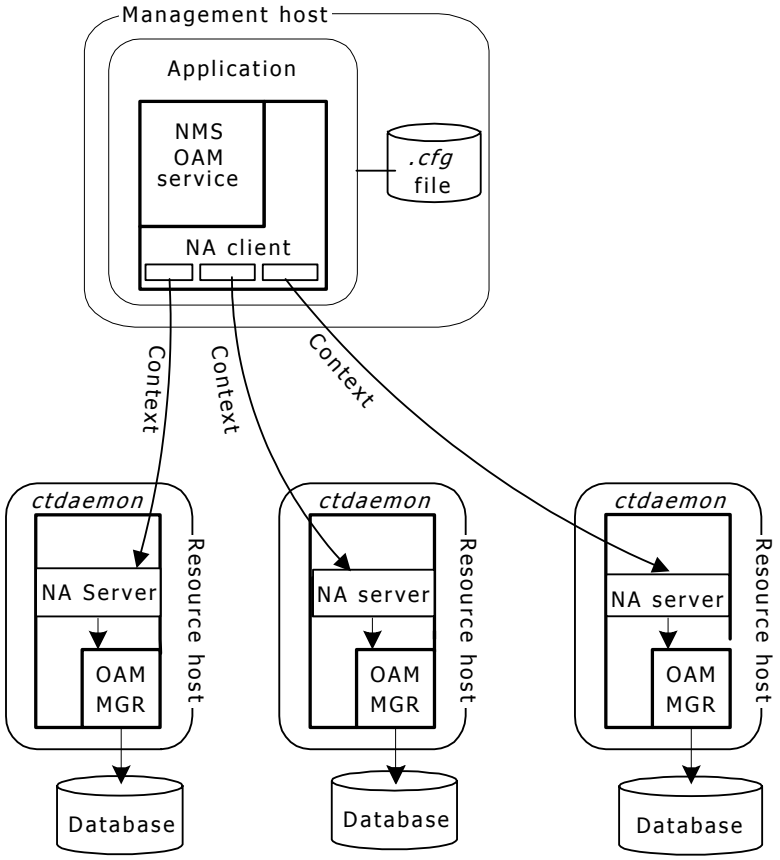



## Multiple-server configuration

Applications on the management host can use NMS OAM on a resource host to configure, start, and manage its resources based on the NMS OAM database on that host.

To specify the resource host to perform an operation on, an application running on the host opens a context on the resource host. An application can open as many contexts to as many resource hosts as needed.

The following illustration shows contexts in multiple-server configurations:




## Hot Swap overview

NMS OAM Hot Swap functionality is used with CompactPCI Hot Swap-compliant boards on Windows and UNIX systems.

A Hot Swap-compliant board has a switch built into the ejector handle and a front panel Hot Swap LED. When the board is inserted, the switch signals that the board is fully seated (with the handle closed) and the supporting software can be initialized. When the board is extracted, the switch signals that the operator is beginning to extract the board and that software disconnection should be initiated.

When lit, the Hot Swap LED indicates that software disconnection is complete and the board can be safely extracted. The operator can open the handle the rest of the way and extract the board.

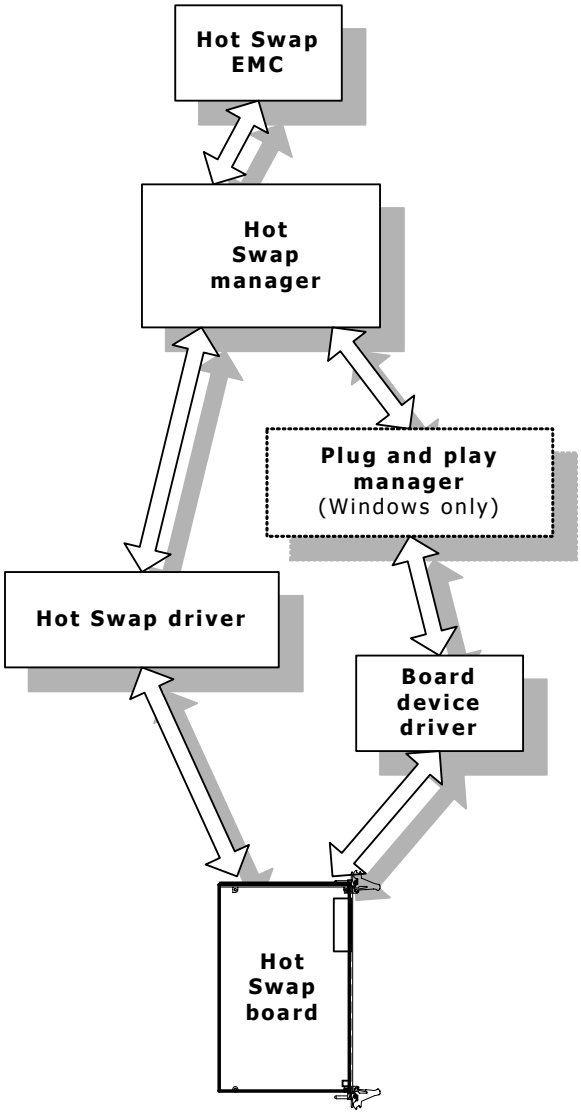
Separate instances of the Hot Swap software reside on each host running NMS OAM that supports Hot Swap. The Hot Swap software on a resource host tracks only boards installed in that host.

## Hot Swap architecture

The following software components make up NMS OAM Hot Swap. These components are installed on each host supporting Hot Swap:

- Hot Swap EMC
- Hot Swap manager
- Hot Swap driver (under Solaris, a set of board device drivers; under Windows, a plug and play manager; under Linux, part of the board device driver)

The following illustration shows the Hot Swap components:



## Hot Swap EMC

The glue logic between the Hot Swap manager library and the NMS OAM Supervisor is implemented as a standard NMS OAM extended management component (EMC). The Hot Swap EMC is a managed object in the NMS OAM database. This object allows an application to access Hot Swap information using the NMS OAM service.

## Hot Swap manager

The Hot Swap manager maintains the states of all the Hot Swap boards in the resource host. Under Windows, the Hot Swap manager is a native service.

The Hot Swap manager coordinates the insertion and extraction of boards and informs applications through the NMS OAM service of the corresponding state of the Hot Swap boards. The NMS OAM service uses the Hot Swap manager to know when to load CompactPCI boards.

The Hot Swap manager also provides the interface between applications and device drivers that monitor Hot Swap boards.

## Hot Swap driver

The Hot Swap driver handles board-independent aspects of interfacing with the Hot Swap hardware. The board device driver handles board-specific Hot Swap duties.

The Hot Swap driver contains different kernel drivers depending on the operating system, as described in the following table:

| Operating system | Kernel drivers |
|---|---|
| Windows | • Hot Swap hardware driver<br>• HS Register miniport drivers<br>• PCI bus filter driver<br>For details, refer to *Hot Swap driver under Windows* on page 23. |
| Solaris | • PCI BIOS driver<br>• Hot Swap hardware driver<br>• Resource manager driver<br>The Hot Swap driver service coordinates these drivers. For details, refer to *Hot Swap driver under Solaris* on page 23. |
| Linux | Windrvr6<br>The Hot Swap manager communicates with Windrvr6, using the Hot Swap libraries. For details, refer to *Hot Swap driver under Linux* on page 24. |

The Hot Swap driver handles the hardware and the Hot Swap manager handles the software. The following table lists various Hot Swap tasks and the components responsible for each task.

| Task | Component |
|---|---|
| Monitoring HS_CSR | Hot Swap driver |
| Tracking the arrival and departure of Hot Swap capable boards | Hot Swap driver |
| Setting the board's Hot Swap LED | Hot Swap driver |

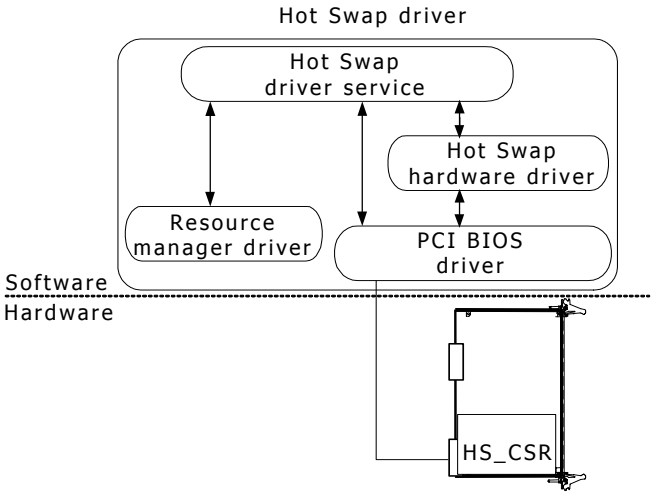
| Task | Component |
|---|---|
| Handling PCI configuration of inserted boards | Plug and play driver (Windows)<br>Hot Swap driver (Solaris and Linux) |
| Handling the de-configuration of any boards being prepared for extraction | Plug and play driver (Windows)<br>Hot Swap driver (Solaris and Linux) |
| Detecting when all channels are closed on the board | Board device driver |

## Hot Swap driver under Windows

Under Windows, you must install the NMS Hot Swap Kit (HSK) for Hot Swap to operate. The HSK is included with Natural Access, but is not installed automatically by the installation program. For more information, refer to the Natural Access installation booklet.

## Hot Swap driver under Solaris

The following illustration shows the Hot Swap driver architecture under Solaris:




The following table describes the Hot Swap Solaris components:

| Driver | Description |
|---|---|
| Resource manager driver | • Implements the interface to the operating system resource database (such as *resmgr*, registry, device tree).<br>• Requests information about the current system configuration, allocated memory, and IRQs.<br>• Updates the resource database in conjunction with insertions and extractions. |
| PCI BIOS driver | • Enables access to the PCI BIOS (or PCI BIOS-like) functions.<br>• Enables PCI configuration space read and write operations. |
| Hot Swap hardware driver | • Implements the interface to the Hot Swap hardware (such as ENUM, on board HS_CSR).<br>• Uses the HSBIOS driver to access PCI configuration space. |

### Hot Swap driver under Linux

Under Linux, the Hot Swap driver is part of the NMS driver packages. After Natural Access is installed, use the following command to set up the Hot Swap driver and HotSwap.emc. This command also starts the Hot Swap manager as a daemon.

The following command configures the Hot Swap driver to load automatically, on subsequent reboots of the host.

```
# /opt/nms/hotswap/bin/hsmgr_startup start
```

The following command starts the Hot Swap manager (as a daemon) on subsequent reboots of the host.

```
# /opt/nms/hotswap/bin/hsmstart start
```

The following command stops the Hot Swap manager:

```
# /opt/nms/hotswap/bin/hsmstart stop
```

The following command unloads and disables automatic loading of the Hot Swap driver on subsequent host booting:

```
# /opt/nms/hotswap/bin/hsmgr_startup stop
```

# 4 Processing NMS OAM service events

## NMS OAM event overview

NMS OAM events are generated by:

- The NMS OAM Supervisor, in response to an NMS OAM asynchronous function (such as **oamStartBoard**, **oamStopBoard**, or **oamTestBoard**), or when lower-level events or errors occur.
- The Hot Swap EMC, when Hot Swap state changes or other events or errors occur.
- The Clock Management EMC, when events or errors occur.
- Board plug-ins, when events or errors occur.
- Other registered applications, using **oamAlertNotify**.

To receive events from the NMS OAM service and from installed EMCs such as Hot Swap and Clock Management, a client application must register with the NMS OAM service using **oamAlertRegister**. See *Receiving NMS OAM alert notifications* on page 33.

NMS OAM events are encapsulated within the standard CTA event structure. In NMS OAM events, the CTA event buffer field points to an OAM_MSG structure that contains more information about the event. After processing NMS OAM events, applications must free the Natural Access event buffers as described in *Event buffer management* on page 29.

## CTA event structure

NMS OAM events arrive encapsulated in the standard CTA_EVENT data structure (defined in *ctadef.h*):

```
typedef struct CTA_EVENT
{
    DWORD    id;             /* Event code (and source service id)      */
    CTAHD    ctahd;          /* Context handle                          */
    DWORD    timestamp;      /* Timestamp                               */
    DWORD    userid;         /* User id (defined by ctaCreateContext)   */
    DWORD    size;           /* Size of buffer if buffer != NULL        */
    void     *buffer;        /* Buffer pointer                          */
    DWORD    value;          /* Event status or event-specific data     */
    DWORD    objHd;          /* Service client side object handle       */
} CTA_EVENT;
```

This structure, returned by **ctaWaitEvent**, informs the application which event occurred on which context, and includes additional information specific to the event.

The CTA_EVENT structure contains the following fields:

| Field | Description |
|---|---|
| id | Event code defined in the library header file. For example: OAMEVN_SOMETHING_HAPPENED.<br>For a list of NMS OAM event codes, see *NMS OAM events* on page 151. |
| ctahd | Context handle returned from **ctaCreateContext**. |
| timestamp | Time when the event was created in milliseconds since midnight, January 1, 1970, modulo 49 days.<br>An application can decode the timestamp using **ctaGetTimeStamp**. |
| userid | User-supplied identifier. This field is unaltered by Natural Access and facilitates asynchronous programming. Its purpose is to correlate a context with an application object/context when events occur. |
| size | Size (bytes) of the area to which the buffer field points. This field also indicates whether a data buffer associated with the event must be freed using **ctaFreeBuffer**. If the buffer is NULL, this field can hold an event-specific value.<br>For NMS OAM events, size indicates the size of the OAM_MSG structure. |
| buffer | Pointer to data returned with the event. The field contains an application process address and the event's size field contains the actual size of the buffer.<br>For NMS OAM events, the buffer points to an OAM_MSG structure.<br>**Note:** After processing an NMS OAM event, the application must free the event buffer for Natural Access. For more information, see *Event buffer management* on page 29. |
| value | Event-specific value. If the event is a DONE event (an event indicating that a function was successfully invoked, such as OAMEVN_STARTBOARD_DONE), this field holds a reason code indicating the actual results of the function.<br>For more information, see *NMS OAM reason codes* on page 155. |
| objHd | The service client-side object handle. |

## NMS OAM event structure

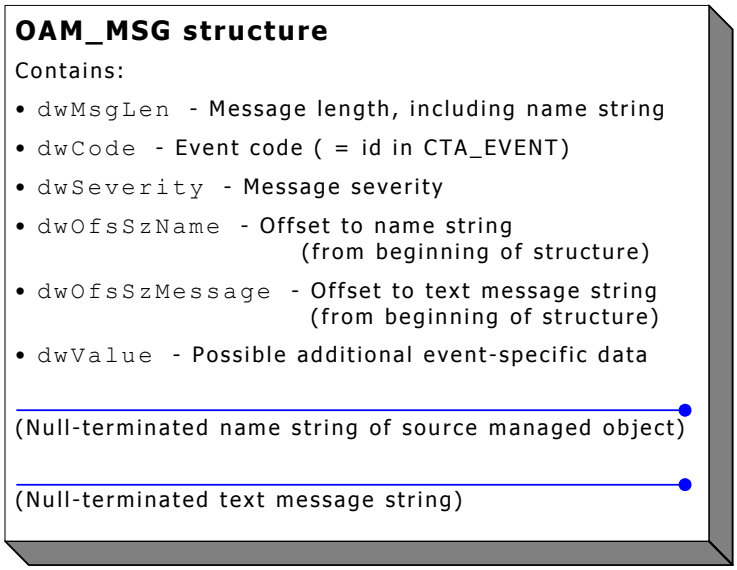
The buffer field in the CTA_EVENT structure points to an OAM_MSG structure defined in *oamdef.h.* The structure contains more information about the event:

```
typedef struct oam_msg_tag
{
    DWORD dwMsgLen;       // Message length, including appended      //
                          // name & message strings                  //
    DWORD dwCode;         // Message code                            //
    DWORD dwSeverity;     // Message severity                        //
    DWORD dwOfsSzName;    // Offset to name string of source         //
                          // managed object                          //
    DWORD dwOfsSzMessage; // Offset to text message string           //
    DWORD dwValue;        // Possible additional event-specific data //
    // (String data is appended here) //
} OAM_MSG;
```

The application can use the OAM_IS_OAM_EVENT macro (defined in *oamdef.h*) to determine if an incoming event is an NMS OAM event.

As shown in the following illustration, the OAM_MSG structure contains six DWORD fields and two strings that contain:

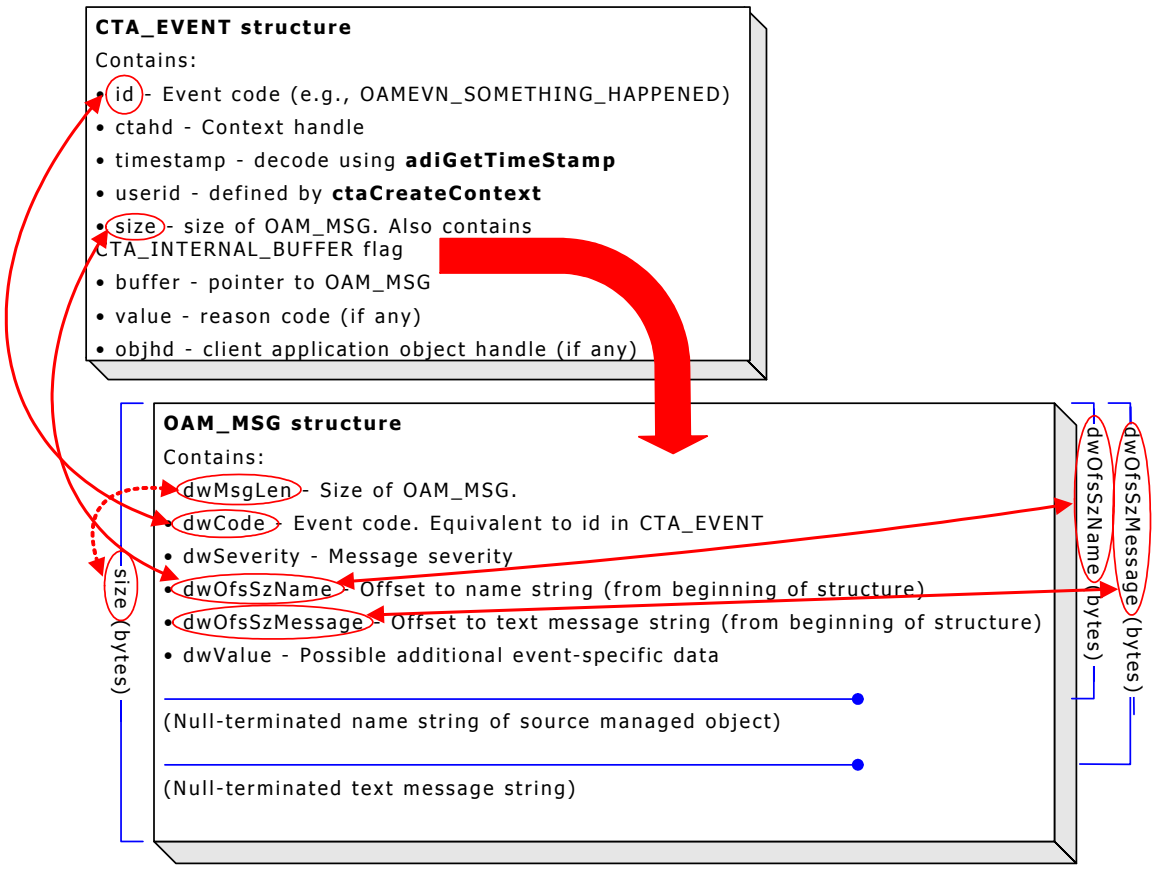
- The name of the managed object generating the event, and
- Any additional text messages.



The OAM_MSG structure fields are described in the following table:

| Field | Description |
|---|---|
| dwMsgLen | The length of the message appended to the end of the structure, including the name string and the text string. |
| dwCode | Event code (always equivalent to id in the CTA_EVENT structure). |
| dwSeverity | Severity of the message. Possible values are:<br>CTA_TRACE_SEVERITY_INFO - Message is informational.<br>CTA_TRACE_SEVERITY_WARNING - Message is a warning.<br>CTA_TRACE_SEVERITY_ERROR - Message is an error. |
| dwOfsSzName | Offset to the first byte of the name string, from beginning of OAM_MSG structure. |
| dwOfsSzMessage | Offset to the first byte of the text message string, from beginning of OAM_MSG structure. |
| dwValue | Some events place additional data here. For more information, refer to the board documentation. |
| (name string) | The name of the managed object associated with the message. Null-terminated. |
| (text message string) | Free-form text giving additional information associated with the message. Null-terminated. |

The following illustration shows the relationship between CTA_EVENT and OAM_MSG, and the fields they contain:



## Event buffer management

To avoid memory leaks, the client application must free the Natural Access event buffer after processing an NMS OAM event. When the buffer must be freed for Natural Access, the flag CTA_INTERNAL_BUFFER is set in the size field. Use **ctaFreeBuffer** to free the event buffer.

**Note:** Mask out the flag before using the size buffer.

If the CTA_INTERNAL_BUFFER is not set, the application does not need to free the event buffer, even if the buffer contains data.

For more information about event buffers, refer to the *Natural Access Developer's Reference Manual*.

### Example

The following code demonstrates how to detect and process an NMS OAM event:

```
////////////////////////////////////////////////////////////////////////////
// monitor
//
// Loops monitoring the OA&M system.

static const char *getText( DWORD dwCode );

static void monitor()
{
  DWORD dwErr;
  for (;;)
  {
     // Retrieve an event from the event queue.
     CTA_EVENT cta_event;
     dwErr = ctaWaitEvent( s_ctaqueuehd, &cta_event, 1000 );
     if (dwErr != SUCCESS)
     {
        printf( "Error waiting for event, %s.\n", getText(dwErr) );
        return;
     }

     // Check if buffer is owned by CTA and must be freed by us below.
     bool const bCtaOwnsBuf = cta_event.buffer && (cta_event.size &
                                                   CTA_INTERNAL_BUFFER);
     if (bCtaOwnsBuf)
         cta_event.size &= ~CTA_INTERNAL_BUFFER; // clear flag from size

     // Process the event.
     if (cta_event.id == CTAEVN_WAIT_TIMEOUT)
     {
      . . .
     }
     else if (OAM_IS_OAM_EVENT(cta_event.id)) // if it's an NMS OAM event
     {
        if (!cta_event.buffer)
        {
           printf( "Error, NMS OAM event has NULL buffer.\n" );
           continue;
        }
        const OAM_MSG * const pOamMsg = (OAM_MSG *)cta_event.buffer;
        const char * const pszName = (char*)pOamMsg + pOamMsg->dwOfsSzName;
        const char * const pszText = (char*)pOamMsg + pOamMsg->dwOfsSzMessage;

        const char *pszSeverity;
        switch (pOamMsg->dwSeverity)
        {
          case CTA_TRACE_SEVERITY_INFO:         pszSeverity = "INFO"; break;
          case CTA_TRACE_SEVERITY_WARNING:      pszSeverity = "WARNING"; break;
          default: /* CTA_TRACE_SEVERITY_ERROR */ pszSeverity = "ERROR"; break;
```

```
        }
        time_t t;
        unsigned int nMsecs = 0;
        dwErr = ctaGetTimeStamp( cta_event.timestamp, (unsigned long*)&t,
                                                        &nMsecs );
        if (dwErr != SUCCESS)
           printf( "Error converting timestamp: %s\n", getText(dwErr) );

        // Get the local date and time
        char szDateTime[80];
        strcpy(szDateTime, ctime(&t));
        // Truncate the year and trailing carriage return.
        szDateTime[strlen(szDateTime) - 6] = '\0';
        printf( "%s - %s", szDateTime, getText(pOamMsg->dwCode) );
        if (CTA_IS_DONE_EVENT(pOamMsg->dwCode)) //If DONEevent,show reason code
            printf( " %s", getText(cta_event.value) );
        printf( " %s", pszSeverity );
        printf( " \"%s\"\n%s\n", pszName, pszText );
     }
     else    // else it's a non-NMS OAM event
         printf( "%s\n", getText(cta_event.id) );

     if (bCtaOwnsBuf)
         ctaFreeBuffer( cta_event.buffer ); // our reponsibility to free
  } // end for(;;)
}

////////////////////////////////////////////////////////////////////////////
// getText
//
// Helper function that uses ctaGetText() to format the text for a numeric code
// (error, event, reason, or command code) and returns a pointer to that text.
// This is actually a pointer to a local static buffer.  It is not thread-safe
// and will be overwritten by subsequent calls.
// If ctaGetText() fails, formats the numeric code value in hex.
//
// dwCode [in] is the numeric code.
//
// Returns a pointer to the text.

static const char *getText( DWORD dwCode )
{
    DWORD const dwBufSize = 80;
    static char szBuf[ dwBufSize ];

    if (SUCCESS != ctaGetText( s_ctahd, dwCode, szBuf, dwBufSize ))
        sprintf( szBuf, "0x%08lx", dwCode );

    return szBuf;
}
```

# 5 Initializing the NMS OAM service

## Initializing NMS OAM client applications

To access NMS OAM service functions, a client application must:

1. Initialize Natural Access.
2. Create event queues.
3. Create contexts.
4. Open NMS OAM service instances on the context, or attach to an existing shared context if one exists.
5. Register for NMS OAM alert notification messages (if the application needs to receive NMS OAM events).

### Initializing Natural Access

Use **ctaInitialize** to initialize Natural Access applications. When you initialize Natural Access applications you register the services that are available to the application.

Applications must invoke **ctaInitialize** before calling any other Natural Access functions. For general information about **ctaInitialize**, refer to the *Natural Access Developer's Reference Manual*.

#### Registering the NMS OAM service

You can register services in one of the following ways:

- Invoke **ctaInitialize** and specify service and service manager names. To use the NMS OAM service, specify OAM as the service name and OAMMGR as the service manager name in the CTA_SERVICE_NAME structure passed in the **ctaInitialize** invocation. The application can use only the services specified in this invocation.
- Invoke **ctaInitialize**, pass NULL for ***svcname***, and pass a user-defined Natural Access configuration file name (usually *cta.cfg*). **ctaInitialize** looks in the [ctasys] header section of the configuration file for the service and service manager names. To use the NMS OAM service, OAM and OAMMGR must appear in this section, as follows:

  ```
  Service = oam, oammgr
  ```

- Initialize the application on a local host. *ctdaemon* automatically provides access to the services specified in the *cta.cfg*. If an application specifies a service in the **ctaIntialize** invocation that is not available on the host, an error is returned.

## Creating event queues

After initializing Natural Access, invoke **ctaCreateQueue** to create one or more event queues. In the function invocation, specify the service managers to be attached to each queue.

## Creating contexts

Invoke **ctaCreateContext** or **ctaCreateContextEx** (for shared contexts) to create a context. Provide the queue handle (***ctaqueuehd***) that was returned from **ctaCreateQueue**. All events for services on the context are received in the specified event queue.

Use the ***descriptor*** string argument passed with **ctaCreateContext** or **ctaCreateContextEx** to specify the resource host on which the context will be created. ***descriptor*** has the following format:

[cta://]***host_address***[:***port***]

**ctaCreateContext** and **ctaCreateContextEx** each return a context handle (***ctahd***). The context handle is supplied by the application when invoking service functions. Events communicated back to the application are also associated with the context.

Refer to the *Natural Access Developer's Reference Manual* for details on the programming models created by the use of contexts and event queues.

## Opening services

Invoke **ctaOpenServices** to open services on a context. When opening a service on a context, you must specify the service and the service manager, and the resources to attach to the context. Specify this information in substructures of the CTA_SERVICE_DESC structure that you pass in the function invocation. NMS OAM functions are available only to contexts on which the NMS OAM service is opened.

For more information, refer to the *Natural Access Developer's Reference Manual*.

To receive events from the NMS OAM service and from installed EMCs (such as Hot Swap and Clock Management), a client application must register with the NMS OAM service using **oamAlertNotify**.

## Receiving NMS OAM alert notifications

To receive NMS OAM events from a resource host, a client application must register with the Supervisor on the resource host to receive alert notification messages. An application registered on a given resource host receives all NMS OAM-generated events for all applications registered with that host, regardless of which client application triggered which event. These events include board-level tracing information.

A client application registers for alert notification using **oamAlertRegister**. With this function invocation, the client application passes the handle to the NMS OAM service. The application is immediately registered for alert notification. The application must register separately with each resource host to receive events from that host.

To unregister from alert notifications, a client application uses **oamAlertUnregister**. Pass the handle to the NMS OAM service when invoking this function. The application is immediately unregistered.

**Note:** Rather than unregistering, a registered client application can use **ctaSetEventSources** to mask out NMS OAM events. For more information, refer to the *Natural Access Developer's Reference Manual*.

## Logging startup events

NMS OAM automatically maintains the following event logs on each host:

| File name | Description |
|---|---|
| *startup.log* | A list of all NMS OAM events that occurred when the host started. This file is rewritten whenever the host starts. It is closed after the host starts. |
| *oam.rpt* | Low-level report information generated whenever a board is started. Information about each board is appended to the file when the board is started. The file is rewritten when the host is started. |

On the host, the files can be found in the following locations:

- Windows: *\nms\oam\log*
- UNIX: */opt/nms/oam/log*

# 6 Managing configuration data

## Accessing configuration data

Within an NMS OAM configuration database on a resource host, a record of configuration information exists for each managed object in the host. The Supervisor component of the NMS OAM service manages the database. Client applications can add, delete, or modify this information using keywords.

**Note:** The *oamsys* and *oamcfg* utilities included with NMS OAM provide an alternate way to access and modify configuration information in the managed objects in the NMS OAM database. For more information about NMS OAM utilities, refer to the *NMS OAM System User's Manual*.

To access configuration data for a managed object, a client application:

1. Opens the managed object.
2. Retrieves keyword values or qualifiers, or sets values as necessary.
3. Closes the managed object.

### Opening a managed object for editing

To access configuration data for a managed object, a client application opens the object using **oamOpenObject**. With this function invocation, the application supplies:

- Context handle to the host where the managed object is located.
- Name of the managed object. This name is not case sensitive.
- Opening mode. Valid modes are:

| Mode | Description |
|---|---|
| OAM_READONLY | Opens the object for reading only. The application cannot change keyword values.<br>A managed object can be opened in read-only mode even if it is currently opened in read-only mode by another application. |
| OAM_READWRITE | Opens the object for reading and writing. The application can change the values of read and write keywords.<br>A managed object cannot be opened in this mode if any other client application has it currently opened. |

If a client application attempts to open a managed object for writing, and the object is currently opened by another application, **oamOpenObject** fails and OAMERR_ACCESS_DENIED is returned to the application.

An open managed object cannot be deleted by any application other than the application that opened the object.

If the object-opening attempt succeeds, **oamOpenObject** returns a handle to the object, to be used with subsequent editing functions.

## Retrieving keywords and qualifiers

All configuration data in the configuration database is represented and communicated as keyword name and value pairs:

***keyword*=*value***

Example:

```
AutoStart=YES
```

This approach is used for every configurable feature of every managed component, including boards, extended management components such as Hot Swap, and the Supervisor itself.

Each keyword has several attributes called qualifiers. For example, the qualifier Type indicates the type of value it accepts (for example, Integer or String). The qualifier ReadOnly indicates that a keyword is read only. Qualifiers are not case sensitive.

Once an object has been opened using **oamOpenObject**, a client application can retrieve values for the object's keywords using **oamGetKeyword**. **oamGetKeyword** takes a keyword name. This name is not case sensitive. If the function invocation is successful, **oamGetKeyword** returns a buffer containing the string value of the keyword.

A client application can retrieve the value of a keyword qualifier using **oamGetQualifier**. If the function invocation is successful, **oamGetQualifier** returns a buffer containing the value of the specified keyword qualifier.

## Setting a keyword value

Once an object has been opened using **oamOpenObject**, a client application can set the value of a keyword using **oamSetKeyword**. The object must have been opened in OAM_READWRITE mode. If the function invocation is successful, **oamSetKeyword** returns SUCCESS.

Most keyword settings are committed when you close the managed object. However, actual operation may not be affected until you restart the board. For details, see the board-specific documentation.

## Closing a managed object

To close a managed object, a client application invokes **oamCloseObject**. If the function invocation is successful, the object is closed. If the object was opened in read and write mode, any changes are validated and saved. If the object being closed was opened in read-only mode, it remains open from the point of view of any other application.

## Working with keywords and keyword values

Keywords are not case sensitive. All values are strings, or strings that represent integers. An integer keyword can have a fixed numeric range of legal values. A string keyword can support a fixed set of legal values or accept any string.

There are eight different keyword types. The type of a particular keyword can be determined by retrieving its Type qualifier. For more information, refer to *Keyword qualifiers* on page 38. The following table describes each keyword type:

| This keyword type... | Indicates that the keyword represents... |
|---|---|
| Integer | A string representing an integer value. For example: 12 |
| String | A string value. |
| Array | An array of multiple values. |
| Struct | A structure containing multiple named values (elements). |
| StructAndArray | Both a structure with multiple elements and an array of values. |
| Filename | A file name (string). For example: *readme.txt* |
| Object | A managed object name (string). For example: Supervisor |
| IPAddress | An IP address (string). For example: 255:255:255:0 |

## Keyword qualifiers

The following table lists the qualifier types:

| Qualifier | Description | Valid values |
|---|---|---|
| Type | Type of keyword value (see Array keywords). | Integer, String, Array, Struct, StructAndArray, Filename, Object, IPAddress |
| ReadOnly | Indicates if keyword value is read only. | No indicates keyword value is read/write. Yes or the absence of a value indicates that the keyword value is read-only. |
| Base | Indicates the mathematical base of an integer. For example, 16 indicates a hexadecimal number. | Any integer |
| Min | If type is Integer, indicates the minimum allowed value. | Any integer |
| Max | If type is Integer, indicates the maximum allowed value. | Any integer |
| Choices | Represents an array containing a list of valid values for the keyword. For read and write keywords only. | Array |
| Choices.Count | Indicates the number of items in array Choices. Zero-based. | Any positive integer |
| Choices[***x***] | Represents value of item ***x*** in array Choices.<br>***x*** is an integer between 0 and the value of Choices.Count - 1. | Any string |
| Description | Description of the keyword. | Any string (80 characters or less) |
| Keywords | Represents an array containing a list of elements contained within a Struct (see *Array keywords* on page 39). | Array |
| Keywords.Count | Indicates the number of items in array Keywords. Zero-based. | Any positive integer |
| Keywords[***x***] | Represents keyword ***x*** in array keyword Keywords.<br>***x*** is an integer between 0 and the value of Keywords.Count - 1. | Any keyword |

## Basic keywords

Basic keyword types String, Integer, Filename, Object, and Address all represent string values. For an Integer keyword, the Min and Max qualifiers indicate a minimum and maximum allowed value. The Base qualifier indicates the mathematical base of the value (for example, base 16 for hexadecimal numbers). For all other basic keyword types, the Choices qualifier indicates a list of valid values.

## Array keywords

Certain values are organized into arrays. Each array is represented by a keyword of type Array. For an Array keyword, the following syntax is used:

| Syntax | Description |
|---|---|
| ***keyword***[***item#***] | Represents item# in array keyword. Arrays are zero based. For example, if the first item assigned to array keyword DSPFile is file1.dsp, and the second item in the array is file2.dsp, then DSPFile[0]=file1.dsp and DSPFile[1]=file2.dsp. |
| ***keyword***.Count | Read-only keyword indicating the number of items assigned to array ***keyword***. For example, if there are five items assigned to array keyword DSPFile, then DSPFile.Count=5. |

If ***keyword***.Count is 0, the array is empty. You can set an item of an empty array to a value, implicitly allocating space within the database record. For example, if the default configuration for a board lists no TCP files, then invoking **oamGetKeyword** with the keyword TCPFiles.Count returns 0. However, you can set TCPFiles[0] to a value (such as nocc). Then when you invoke **oamGetKeyword** again for the keyword TCPFiles.Count, the function returns 1.

The Array keyword itself without an index does not represent a value. For example, if **oamGetKeyword** is performed for keyword DSPFile (without an index), the error OAMERR_NOT_FOUND is returned.

Set each array item individually. The utility *oamsys* translates a line from a board keyword file into separate assignments for each item. For example, if the board keyword file contains the following entry:

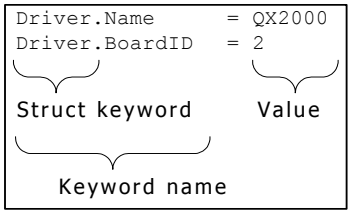
```
Resource[1].DSPs = 1 2 3
```

*oamsys* translates this entry into the following items:

```
Resource[1].DSPs[0] = 1
Resource[1].DSPs[1] = 2
Resource[1].DSPs[2] = 3
```

## Struct keywords

Struct keywords are similar to C language structures. A struct is a group of related named values (elements) under a common name. The fully qualified keyword name for each element in a struct consists of the struct name, followed by a period (.) and then the element name as shown in the following illustration. Within NMS OAM, the fully qualified keyword name for an element is always used to refer to the element.




Structs can contain structs. In the following example, struct Clocking contains structs HBus and MVIP:

```
Clocking.HBus.ClockMode = MASTER_A
Clocking.HBus.AutoFallBack = YES
Clocking.MVIP.ClockRef = SEC8K
Clocking.MVIP.AutoFallBack = NO
```

Structs can contain arrays. For example, struct NetworkInterface contains array T1E1[***x***]. This is specified as NetworkInterface.T1E1[***x***].

An array can contain multiple instances of a struct, each containing an identical set of elements. For example, Resource[***x***] is an array containing multiple instances of struct Resource. Resource contains several elements, some of which are arrays:

```
Resource[0].Name = RSC1
Resource[0].Size = 120
Resource[0].FileName[0] = myfile.foo
Resource[0].FileName[1] = myfile2.foo
Resource[0].SpanEnable=AUTO
Resource[1].Name = RSC1
Resource[1].Size = 60
Resource[1].FileName[0] = myfile.foo
Resource[1].SpanEnable=AUTO
```

For any Struct keyword, the qualifier Keywords represents an array containing a list of elements contained within the Struct keyword. For example, for the Resource[***x***] keyword:

- The qualifier Keywords[1] yields "Name"
- The qualifier Keywords[2] yields "Size"
- The qualifier Keywords[3] yields "FileName"
- The qualifier Keywords[4] yields "SpanEnable"

The qualifier Keywords.Count indicates the number of elements in the Struct keyword. This value is zero based. For example, for the Resource[***x***] keyword, Keywords.Count yields 3.

A Struct keyword without an element does not represent a value. If **oamGetKeyword** is performed for a keyword that is of type Struct, the error OAMERR_NOT_FOUND is returned.

## StructAndArray keywords

A StructAndArray keyword represents both a struct and an array. As a hypothetical example, struct Trunks contains one element, GlobalParm, with a global parameter string value. Also, array Trunks[***x***] contains a specific parameter string, SpecificParm, for trunk ***x***.

Just as with Array keywords, the keyword name ***xxx***.Count indicates the number of items assigned to array keyword ***xxx***. Also, as with Struct keywords, the qualifier Keywords for a StructAndArray keyword represents an array containing a list of keywords within the struct. The qualifier Keywords.Count indicates the number of elements in the struct.

The StructAndArray keyword itself does not represent a value. If **oamGetKeyword** is performed for a keyword that is of type StructAndArray, the error OAMERR_NOT_FOUND is returned.

## Enumerating keywords

An application can discover the set of keywords in a database record. By querying the type and certain qualifiers of each keyword, the application can obtain sufficient information to further process the keyword. If a keyword represents a data structure (such as a struct or array), the application can determine the elements within the data structure. By executing this process recursively, all values of all keywords of a managed object can be discovered. This process is called keyword enumeration.

This topic describes:

- Enumerating the top-level keywords
- Enumerating array keywords
- Enumerating struct keywords
- Enumerating StructAndArray keywords

The *oaminfo* demonstration program included with NMS OAM illustrates the keyword enumeration process. See Demonstration program for more information about *oaminfo*.

### Enumerating the top-level keywords

To enumerate the keywords for a managed object, first retrieve the top-level keywords. Then explore each keyword to determine the subkeywords it contains. Then explore those subkeywords, and so on.

All top-level keywords for a managed object are contained within a single struct with no name. To retrieve the top-level keywords for a managed object:

1. Get the number of elements in the no-name struct. To do so, invoke **oamGetQualifier**, specifying `""` as the keyword, and Keywords.Count as the attribute to retrieve. **oamGetQualifier** returns the number of elements in the struct.
2. To determine the names of each of the elements within the struct, invoke **oamGetQualifier** repeatedly, each time specifying `""` as the keyword, and Keywords[***x***] as the attribute to retrieve. **oamGetQualifier** returns a single element with each invocation.

## Enumerating array keywords

To determine how many items are contained in an array, use **oamGetKeyword** to get the value of keyword ***keyword***.Count. For example, to determine how many items are in array DSPfile, invoke **oamGetKeyword** as follows:

```
oamGetKeyword( hObj, "DSPfile.Count", info, sizeof(info) );
sscanf( info, "%d",numElements )
```

You can then get each specific value ***x*** by invoking **oamGetKeyword** on ***keyword***[***x***]:

```
oamGetKeyword( hObj, "DSPfile[1]", info, sizeof(info) );
```

If the returned value is a keyword, you can determine its Type attribute and perform further analysis and processing of the keyword.

## Enumerating struct keywords

To determine how many elements are contained within a struct, invoke **oamGetQualifier** on the Keywords.Count qualifier for the Struct keyword. For example, to get the number of keywords in struct Driver, use **oamGetQualifier** as follows:

```
oamGetQualifier( hObj, "Driver", "Keywords.Count", info, sizeof(info) );
sscanf( info, "%d",numElements )
```

To then determine the names of each of the elements within the struct, use the qualifier Keywords[x]. For example:

```
for ( i  = 0; i < numFields; i++ )
{
          char field[ 100 ];

          sprintf( field, "Keywords[%d]", i );
          oamGetQualifier(( ...., "Driver", field,  ....  );
}
```

The attribute string returned is in the form: keyword.keyword (for example, "Driver.Name"). By subsequently determining the Type attribute of the returned keyword name, you can perform further analysis and processing of the contents of the struct.

## Enumerating StructAndArray keywords

To perform keyword enumeration for a StructAndArray keyword, perform keyword enumeration treating it as an Array keyword (as described in *Array keywords* on page 39). This process enumerates the elements in the array represented by the keyword. Perform keyword enumeration treating it as a Struct keyword to enumerate the elements in the struct the keyword represents. By subsequently determining the Type attributes of each returned keyword name, you can perform further analysis and processing of the contents of the structs or arrays.

## Importing and exporting configuration data

You can export the contents of the configuration database to a file, and then import the file into the configuration database on another host. Use this feature to set up multiple identical resource hosts.

This topic describes how to export and import configurations using the NMS OAM service. You can also perform these functions using the *oamcfg* utility. For more information about *oamcfg*, refer to the *NMS OAM System User's Manual*.

### Exporting a configuration

To export the contents of a configuration database:

- All client applications must close any managed objects currently open for writing. Otherwise, any pending changes are not included in the exported data.
- A client application invokes **oamConfigExport**, passing it an output file name.

**oamConfigExport** exports a snapshot of the entire configuration to the specified output file. Do not modify the output file because it is used only by the NMS OAM service.

### Importing a configuration

To import the contents of a configuration database from a file:

- Client applications close all open managed objects.

  This is necessary because the current configuration is completely overwritten and all current data is lost.
- A client application invokes **oamConfigImport**, passing it the name of a file exported as described in *Exporting a configuration* on page 43.

**oamConfigImport** imports the entire configuration from the specified input file. The current configuration is lost and is replaced by the new configuration. All plug-ins are restarted.

**Note:** Applications registered for alert notification before the configuration is imported remain registered after the import is complete.

# 7 Managing boards

## Identifying boards

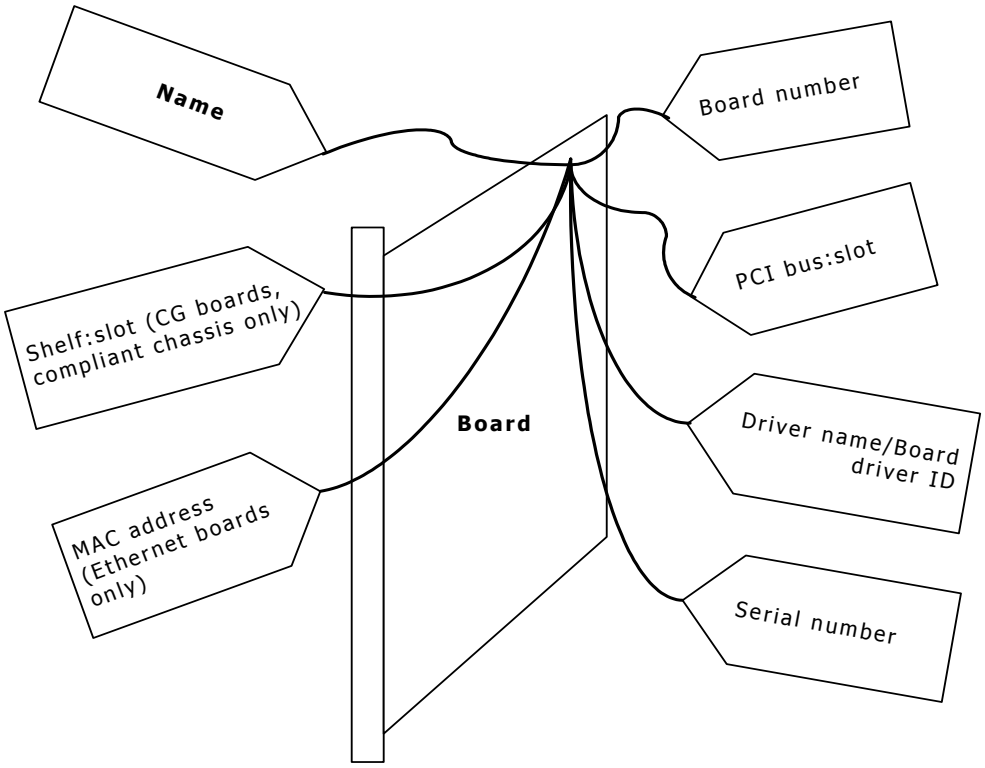
Within NMS OAM, each board is referenced using the following identifiers:

| Identifier | Description |
|---|---|
| Board name | This is the primary way to identify each board in an NMS OAM resource host.<br>A default board name for each board is created when the board is added to the host. This name can be changed by opening the board managed object and modifying the Name keyword.<br>You can use the board name to gain other information about a board. For details, see *Retrieving board information* on page 47. |
| Board number | This is the typical way to identify a board in most NMS software products. Each board in a chassis has a unique board number.<br>A default board number for each board is created when the board is added to the host. This number can be changed by opening the board managed object and modifying the Number keyword. |
| PCI bus and slot | You can identify a board by its unique PCI bus and slot location. |

The following secondary ID information is also available:

| Identifier | Description |
|---|---|
| Driver name/driver board ID combination | The driver name is unique among all driver names in the resource host. The driver board ID is unique among all boards accessed by a given driver. However, two boards accessed by different drivers can have the same driver board ID. The driver name and driver board ID together make up an ID for the board that is unique within the resource host. |
| Serial number | Factory configured number that may not be present for all boards. |
| Shelf and slot | For CompactPCI CG boards only. Each board has a unique shelf number and slot number. The shelf refers to the backplane (or portion of the backplane) in which the board is installed. Slot refers to the physical slot within the chassis where the board is located.<br>Implementation of shelf and slot differs depending upon the chassis manufacturer and specific hardware settings. Shelf and slot information is available only with CG boards installed in CompactPCI chassis with a bus that complies with PICMG 2.1. Refer to the chassis documentation for more information. |
| MAC address | For boards with Ethernet capability only. Each board has unique MAC addresses, one for each Ethernet controller on the board. |

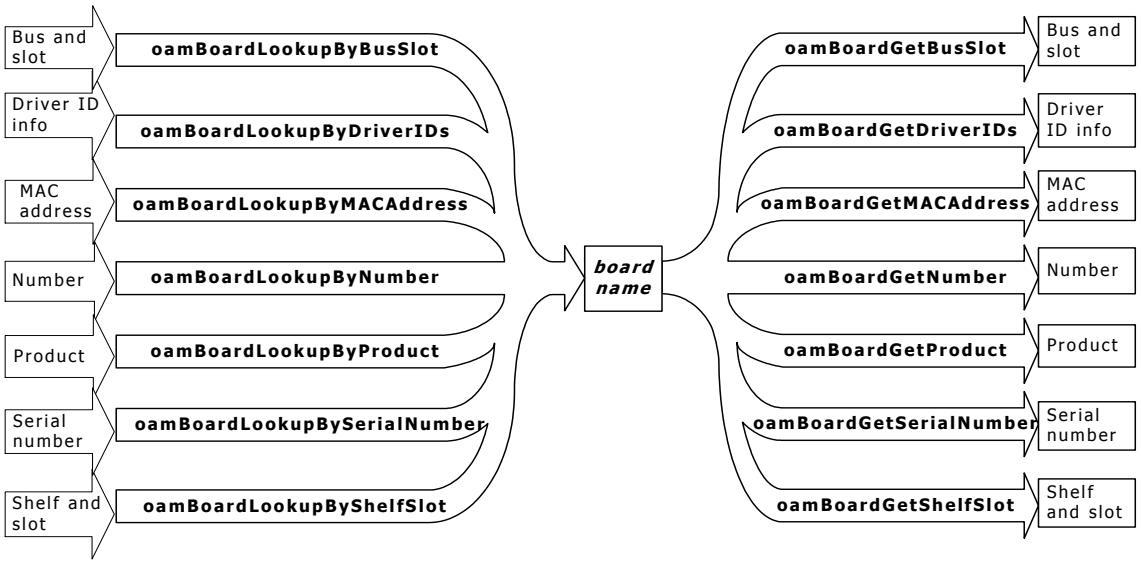
The following illustration shows the identifiers of each board:




The board identifiers are unique to each board in a resource host. However, boards in different resource hosts can have identical identifiers.

## Retrieving board information

The primary way to identify each board to NMS OAM is by its board name. As shown in the following illustration, the NMS OAM service provides several functions to retrieve information about a board, given a board name. If the name is not known, other functions are provided to retrieve the name of a board based on other information about the board.




### Retrieving the name of a board

To retrieve the name of a board from a configuration database, pass any of the following information to the appropriate **oamBoardLookupBy*XXX*** function, along with the context handle to the host where the board is installed and managed by NMS OAM:

- PCI bus and slot
- Driver ID number
- Number
- Serial number
- Product type
- MAC address (Ethernet boards only)
- Shelf and slot (CompactPCI CG boards in PICMG 2.1-compliant chassis only)

**Note:** Not all of this information may be available for a given board.

The function returns a buffer containing the name of the board.

You can retrieve the names of all boards in the database, using **oamBoardEnum**. With the function invocation, pass an index value, starting at 0 and continuing until the function returns OAMERR_NOT_FOUND. A board name is returned with each invocation.

For more information about these functions, refer to *Board information retrieval functions* on page 66.

### Retrieving other board information

Once you retrieve the name of a board, you can retrieve any other information about the board using the appropriate **oamBoardGet*XXX*** function. Pass the name of the board to this function, along with the context handle to the resource host containing the board.

## Managing board information

A client application can use the NMS OAM service to:

- Create board managed objects on a host.
- Delete board managed objects on a host.
- Detect and add boards physically installed in a host. This feature is not supported by all board plug-ins.

### Creating board managed objects

A client application can create managed objects for boards in a resource host, even if the board is not yet physically installed in the resource host. Once a board has a managed object, it can be configured. For more information about configuring managed objects, refer to *Accessing configuration data* on page 35.

To create a board managed object, use **oamCreateBoard**. Pass the following information when invoking the function:

- The context handle to the resource host where the board is installed.
- The product type for the board. When the Supervisor on the resource host starts up, it queries all of the plug-ins installed on the host for the names of all product types that they support. These names are stored in the Supervisor array keyword Products[x].

If a plug-in for a board family is not installed on the host, product type names for the boards in the family are not available, and no boards of the family can be created or accessed.

When the function is invoked, a new managed object is created on the resource host. A record for the managed object is created in the NMS OAM database on the host, containing default keyword settings. A default name and board number are generated for the board:

| Board ID | Default |
|---|---|
| Name | Product type and an integer. For example: CG_6000C_QUAD 0 |
| Number | Next available number greater than or equal to zero (0). |

The board name is passed back in the function call. The name is added to the list of board names in the Supervisor keyword Boards[x].

You can retrieve the board number using any **oamBoardLookupBy*XXX*** function, as described in *Retrieving board information* on page 47.

To change the board name and board number, modify the Name or Number keywords for the board-managed object.

## Deleting board managed objects

A client application can use **oamDeleteBoard** to delete a board managed object from a resource host. To use this function, pass the context handle and the name of the board to delete to the NMS OAM service. Specify NULL to delete all boards. To delete a board, the board must exist as a managed object.

A client application cannot delete a managed object if it is currently opened by another application.

## Detecting installed boards automatically

The NMS OAM service can detect AG and CG boards physically installed in a chassis and create a managed object for each detected board.

To detect and add boards, the client application invokes **oamDetectBoards**. This function detects the boards, generates a board name for each board, and adds the board name and other board information to subkeywords of the Supervisor array DetectedBoards[]. The board name can be changed after a managed object is created for the board.

The following information is collected:

| Information | Stored in keyword... |
|---|---|
| Generated board name | DetectedBoards[x].Name |
| Board product type | DetectedBoards[x].Product |
| PCI bus location | DetectedBoards[x].Location.PCI.Bus |
| PCI slot location | DetectedBoards[x].Location.PCI.Slot |

**Note: oamDetectBoards** does not actually add board configuration information to the configuration database; it only collects board information.

The list of detected boards includes all boards physically installed in the resource host, including boards that already have managed objects.

If you have a chassis with an unusual PCI bus topology (for example, bus number 171 directly follows bus number 0), **oamDetectBoards** may operate more slowly. To speed up operation, create a text file specifying PCI bus numbers to search. For details, refer to the *NMS OAM System User's Manual*.

To create a managed object for a detected board, invoke **oamAddDetectedBoard** with the name of the detected board to add. To add all detected boards at once, specify NULL as the board name.

When **oamAddDetectedBoard** is invoked, a new managed object is created on the resource host for the board. A record for the managed object is created in the NMS OAM database on the host, containing default keyword settings.

**oamAddDetectedBoard** cannot add a detected board to the configuration if the board's name, PCI bus:slot location, or both matches that of another board already listed in the NMS OAM database. If **oamAddDetectedBoard** detects a conflict while attempting to add a single board, it returns OAMERR_ALREADY_EXISTS. If the conflict is detected while **oamAddDetectedBoard** is adding all boards (***szName*** is NULL), it returns OAMEVN_ALERT_INFO for each failed attempt. However, the function returns SUCCESS if at least one board was added successfully to the database.

If you direct **oamAddDetectedBoard** to add all boards (***szName*** is NULL) and **oamDetectBoards** was not previously invoked or has found no boards, **oamAddDetectedBoard** returns OAMERR_NOT_FOUND.

## Starting, stopping, and testing boards

Applications use NMS OAM service functions to start, stop, and test boards.

### Starting boards

A board must exist as a managed object before you can start it using NMS OAM.

Use **oamStartBoard** to start a board. Pass the context handle of the host managing the board and the name of the board to start. To start all boards in a resource host at once, pass NULL as the name. To start boards on multiple resource hosts, invoke this function at least once per host.

**oamStartBoard** is an asynchronous function that sends a return code to the application indicating if it was successfully initiated. If the return value is SUCCESS, the actual results of the board-starting attempt arrive later as events. As the Supervisor attempts to start each board, it returns OAMEVN_STARTBOARD_DONE to the application. The value field returned with the event indicates if the board started successfully:

- OAM_REASON_FINISHED indicates that the board started successfully.
- OAM_REASON_FAILED indicates that the board start attempt failed.

Further information is available in the OAM_MSG message buffer (in its text portion), as well as in other alert notifications generated during the start process.

### Stopping boards

A board must exist as a managed object before you can stop it using NMS OAM.

Use **oamStopBoard** to stop one or more boards on a resource host. Pass the context handle of the host managing the board and the name of the board to stop. To stop all boards in a resource host at once, pass NULL as the name. To stop boards on multiple resource hosts, invoke this function at least once per host.

**oamStopBoard** is an asynchronous function that sends a return code to the application indicating if it was successfully initiated. If the return value is SUCCESS, the results of the board stop attempt arrive later as events. When a request to stop a board is made, the Supervisor generates OAMEVN_STOPBOARD_REQ to inform all registered clients that the board will be stopped in 150 ms. This event allows applications to perform clean up before the board is stopped. As the Supervisor attempts to stop each board, it returns OAMEVN_STOPBOARD_DONE to the application. The value field returned with the event indicates if the board stopped successfully:

- OAM_REASON_FINISHED indicates that the board stopped successfully.
- OAM_REASON_FAILED indicates that the board stop attempt failed.

Further information is available in the OAM_MSG message buffer (text portion), as well as in other alert notifications generated during the board stop attempt.

**Note: oamStopBoard** stops boards regardless of any tasks they are currently performing. To avoid problems, ensure that a board is idle before stopping it.

## Testing boards

Once a CG board is started, a client application can test it and report results. Three levels of tests can be performed on a board:

- A basic test (test level 1).
- An extended test (test level 2). This is the test level automatically run on powerup.
- An exhaustive test (test level 3). Level 3 tests can take time to run.

A client application can use **oamTestBoard** to test one or more CG boards. Pass this function the following information:

- A context handle to the resource host on which NMS OAM is managing the board.
- The name of the board to test. To test all boards in a resource host at once, pass NULL as the name.
- A 32-bit mask describing the test level to perform and the board components to test.

To test boards on multiple resource hosts, invoke **oamTestBoard** at least once per host.

**Note:** Testing can interrupt current board activities. For this reason, do not use a board for any other operations during testing.

### Test level bit mask

For CG boards, the bits in the test level bit mask are interpreted as follows:

- Bit 0 (0x0): if set, **oamTestBoard** scans all testable on-board components and lists them to stdout. If bit 0 is not set, no list is generated.
- Bits 1 (0x2) through 3 (0x8) indicate the level of test to perform:

| Bit | Test |
|---|---|
| 1 (0x2) | Perform test level 1 on specified components. |
| 2 (0x4) | Perform test level 2 on specified components. |
| 3 (0x8) | Perform test level 3 on specified components. |

- Bits 4 (0x10) through 7 (0x80) are reserved.

- Bits 8 (0x100) through 15 (0x8000) indicate the components to test at the test level indicated using bits 1 through 3. The following table lists the components specified by each bit and the tests performed on the components depending upon the test level:

| Bit | Component tested |
|---|---|
| 8 (0x100) | PCI bridges |
| 9 (0x200) | On-board CPU chips |
| 10 (0x400) | DSP chips |
| 11 (0x800) | Framer chips |
| 12 (0x1000) | Ethernet chips |
| 13 (0x2000) | HMIC chips |
| 14 (0x4000) | Memory |
| 15 (0x8000) | HDLC chips |

- Bits 16 (0x10000) through 29 (0x20000000) are not used.
- Bit 30 (0x40000000): if set, all tests immediately stop if an error is reported. If not set, an error does not stop ongoing tests.
- Bit 31 (0x80000000): if set, status information for each component test is reported in OAMEVN_ALERT_INFO events. If not set, status information is not reported.

Here are some examples:

| dwOptions setting | Test performed |
|---|---|
| 0x103 | Test level 3 of PCI bridges only. |
| 0x301 | Test level 1 of PCI bridges, on-board CPU chip, and DSP chips. |
| 0x80000301 | Test level 1 of PCI bridges, on-board CPU, and DSPs. Displays status information and test information for each component tested. |

**oamTestBoard** is an asynchronous function that sends a return code to the application. A return value of SUCCESS indicates if the function was successfully initiated. As the Supervisor attempts to test each board, it returns OAMEVN_TESTBOARD_DONE to the application. The value field returned with the event indicates if the board tested successfully:

- OAM_REASON_FINISHED indicates that the board tested successfully.
- OAM_REASON_FAILED indicates that the board test attempt failed.

Further information is available in the OAM_MSG message buffer (in its text portion), as well as in other alert notifications generated during the test attempt if bit 31 is set in the test bit mask.

# 8 Hot swapping boards

## Hot Swap board insertion and removal processes

The NMS OAM service receives event notification for board insertion or impending board extraction of all Hot Swap boards in the chassis. All Hot Swap events are sent to the queues on which the NMS OAM service is opened.

**Note:** Under Windows, the NMS Hot Swap kit (HSK) must be installed for Hot Swap to operate correctly. The HSK is included with Natural Access, but is not installed automatically. For more information, refer to the Natural Access installation booklet.

This topic describes:

- What happens when you insert Hot Swap compatible boards
- What happens when you remove a Hot Swap compatible board
- Closing resources after extracting the board

For more information about the NMS implementation of Hot Swap and NMS Hot Swap utilities, refer to the *NMS OAM System User's Manual*.

### Hot Swap board insertion

When you insert a Hot Swap compatible board:

1. The Hot Swap driver software assigns resources to the board and initializes the configuration space registers.
2. Once the Hot Swap driver successfully configures the board, it sends notification to the Hot Swap manager.
3. The Hot Swap manager notifies the Hot Swap EMC to prepare the board.
4. The Hot Swap EMC directs the NMS OAM service to initialize the board.
5. The Hot Swap EMC notifies the Hot Swap manager that the board is prepared.
6. Once board initialization is complete, the Hot Swap manager sends an event through the Hot Swap EMC to Natural Access applications, indicating that the board is ready for use by Natural Access applications.

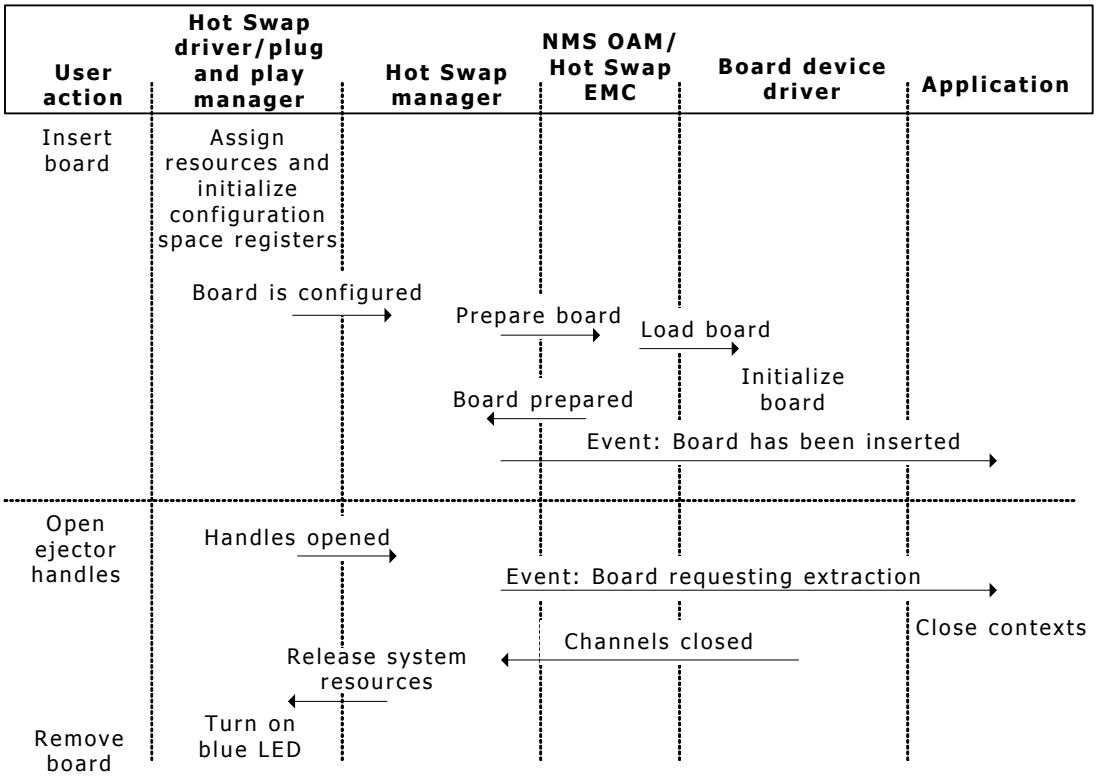
### Board removal

When you open the ejector handles before removing a Hot Swap compatible board:

1. The Hot Swap driver notifies the Hot Swap manager.
2. The Hot Swap manager sends an event through the Hot Swap EMC to Natural Access applications, indicating that the board is requesting extraction.
3. Applications must close all contexts associated with the board. For more information, see *Closing resources* on page 54.
4. The board device driver notifies the Hot Swap manager when all channels are closed.
5. The Hot Swap manager notifies the Hot Swap driver to release system resources allocated to the board (such as address space).

6. The Hot Swap driver de-configures the board and turns on the Hot Swap LED to inform the operator that the board can be safely removed.

The following illustration shows the Natural Access Hot Swap operation:




## Closing resources

The following table lists the actions required by specific Natural Access services to close resources to a board requesting removal:

| For this service... | To release board resources, you must... |
|---|---|
| ADI | Close the service. |
| Digital Trunk Monitor (DTM) | Stop monitoring all trunks on the board. |
| ISDN | Close the service. |
| Natural Call Control (NCC) | Close the service. |
| NaturalConference (CNF) | Close the service |
| NaturalFax (NFX) | Close the service. |
| Switching (SWI) | Close the switch handle. |
| Voice Message (VCE) | Close the service. |

There are no timeouts or other time-related limitations for releasing resources. However, release resources as soon as possible since some operators remove boards within a few seconds of opening the ejector handles.

Once all resources for the board have been released, the Hot Swap LED on the board indicates that the board can be removed.

## Initiating board insertion and extraction in software

You can initiate insertion or extraction by using the:

- *hsmon* utility
- Hot Swap EMC managed object keyword Board.boardname.Command.

These actions correspond to the user actions of opening and closing the board ejector handles.

Use *hsmgr* utility to monitor the Hot Swap manager messages generated when a board is inserted or extracted.

### Using the hsmon utility

To start *hsmon* in console mode, enter the following command:

```
hsmon
```

Once *hsmon* is started, initiate extraction by entering:

```
e PCIbus,slot
```

where ***PCIbus*** and ***slot*** are the PCI bus and slot location of the board.

To initiate insertion, enter the following command:

```
i PCIbus,slot
```

### Using the Board.boardname.Command keyword

To initiate extraction, set the keyword as follows:

```
Board.boardname.Command=Extract
```

where ***boardname*** is the name of the board to extract.

To initiate insertion, set the keyword as follows:

```
Board.boardname.Command=Insert
```

Board.boardname.Command is a write-only keyword. You cannot query it to learn if a board is inserted or extracted. To determine the Hot Swap state of a board, use Board.boardname.State.

For details on Hot Swap EMC managed object keywords, refer to *Hot Swap EMC keyword summary* on page 133.

### Debugging with the Hot Swap manager

For debugging purposes, use the *hsmgr* utility to run the Hot Swap manager in console mode. In this mode, Hot Swap manager messages display as boards are inserted and extracted. To learn more about the *hsmgr* utility and other NMS OAM utilities, refer to the *NMS OAM System User's Manual*.

# Hot Swap state machine

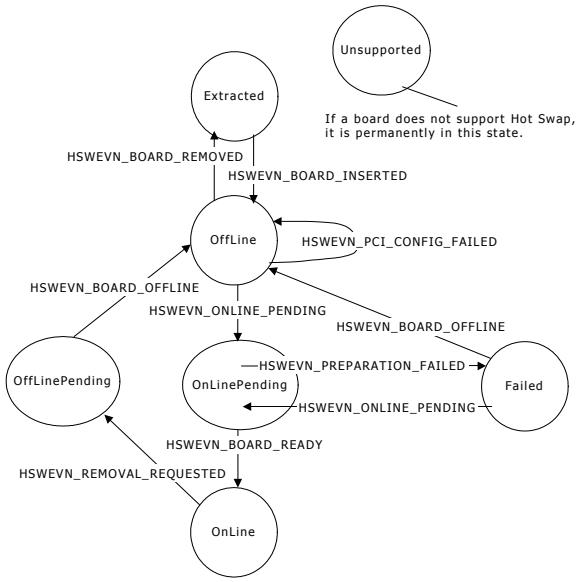
This topic describes Hot Swap states and how to determine the current Hot Swap state.

## Hot Swap states

A managed board can be in any of the following Hot Swap states:

| State | Description |
|---|---|
| Extracted | Board is not present in the chassis. |
| OffLine | Board is present, but is not yet configured on the PCI bus. |
| OnLinePending | Board is being started. |
| Failed | Board start attempt failed. |
| OnLine | Board start attempt succeeded. The board can now support normal operation. |
| OffLinePending | Board ejector handles were opened, and the resource host is preparing to allow the board to be extracted. |
| Unsupported | Board does not support Hot Swap. |

The following illustration shows the Hot Swap state machine:




## Determining the board's Hot Swap state

To determine the current Hot Swap state of a board:

1. Open the Hot Swap EMC managed object using **oamOpenObject**. The managed object name is hotswap.emc.
2. Use **oamGetKeyword** to get the value of the Board.***boardname***.State keyword, where ***boardname*** is the board name of the board (for example, Board.MyBoard.State). The value is a string containing one of the state names (for example, OnLine).

If a board does not support Hot Swap, the returned state is always Unsupported.

For details on Hot Swap EMC managed object keywords, refer to *Hot Swap EMC keyword summary* on page 133.

## Handling surprise extraction

Surprise extraction occurs when the user opens the ejector handles and extracts a board before the Hot Swap LED is turned on. When this occurs, the event notifying closing the service (or context) may not be received.

Contexts that cannot be closed must be ignored. Attempts to use them or close them return CTAERR_INVALID_SEQUENCE.

The following scenario describes how surprise extraction can be detected by a Natural Access application:

1. The application receives HSWEVN_REMOVAL_REQUESTED and begins closing all context or services on the board that generated the event.
2. The application receives CTAEVN_DESTROY_CONTEXT_DONE or CTAEVN_CLOSE_SERVICES_DONE, depending on which function was used to close services on the board. The application saves information for each context if the close event or destroy event is received.
3. HSWEVN_BOARD_OFFLINE, HSWEVN_BOARD_REMOVED, or both occur. The application checks to see if all contexts have received the close or destroy event. If some contexts are in the pending state, the application must not try to use them again.

## Hot Swap in a multiple-host configuration

From the point of view of the client application, Hot Swap operates the same regardless of whether or not the application is located on the same resource host as the board being extracted or inserted. The application receives Hot Swap events if it is registered to receive the NMS OAM alert notification messages, as described in *Receiving NMS OAM alert notifications* on page 33. When a board is being extracted, the application closes all contexts associated with the board and discontinues using the board, as described in *Closing resources* on page 54. When the board is inserted, it is ready for use when the application receives HSWEVN_BOARD_READY.

# 9 Clock management

## H.100/H.110 clocking overview

When multiple boards are connected to the CT bus, you must set up a bus clock to synchronize timing between them. In addition, you can configure alternative (or fallback) clock sources to provide the clock signal if the primary source fails.

This topic provides a brief overview of CT bus clocking, including clock references and clock fallback.

For more information about clocking, refer to the *NMS OAM System User's Manual.*

| | |
|---|---|
| **Caution:** | NMS strongly recommends that you do not mix H.100 or H.110 systems with MVIP systems when using clock fallback. |

Boards in a CT bus system can be configured in any of the following modes:

| Board mode | Description |
|---|---|
| Primary clock master | Drives the primary timing reference for boards connected to the CT bus. It can switch between two specified timing sources to maintain the primary timing reference. However, if both timing references fail, the primary master stops providing a timing source. The secondary master then provides bus synchronization. |
| Secondary clock master | Drives the secondary timing reference. When the primary clock fails, the secondary master continues to drive the secondary clocks using a clock fallback source as its timing reference. |
| Clock slave | References it's timing from the primary clock master and uses the secondary clock master as a fallback source of clock timing. |
| Standalone | Does not reference the primary or secondary master and, consequently, cannot make switch connections to the CT bus. |

Some board models have more flexible and reliable clocking capabilities than others. In a mixed board system, choose the boards with the best capabilities as your primary master and secondary master. To determine which boards to use as masters, refer to the *NMS OAM System User's Manual*.

### Clock references

Boards that act as clock slaves derive their timing from signals driven by the clock masters (primary or secondary). Clock masters can drive the following reference clocks:

- A_CLOCK
- B_CLOCK

Primary clock masters can synchronize their own timing signals from the following sources:

- NETWORK
- NETREF
- OSC

Secondary clock masters are hybrid systems. Their primary timing source must be A_CLOCK or B_CLOCK. Their fallback timing source must be one of the following sources:

- NETWORK
- NETREF
- OSC

## Clock fallback

The CT bus supports a clock fallback mechanism that allows the system to use alternate timing references when one or more sources fail. To enable clock fallback, set the Clocking.HBus.ClockMode keyword to Yes. If you do not enable clock fallback, the application must perform all clocking tasks.

Follow these steps to implement clock fallback:

| Step | Action |
|---|---|
| 1 | Configure a primary clock master to drive the CT bus clock (A_CLOCK or B_CLOCK) based on a network timing reference. All slave boards synchronize their timing through this clock. |
| 2 | Configure a secondary clock master to use the signal from the primary clock to drive the alternate CT bus clock. In other words, if the primary master drives the A_CLOCK, configure the secondary master to drive the B_CLOCK based on the A_CLOCK, or vice versa. |
| 3 | Specify a fallback network timing reference for the secondary clock master to use if the primary clock master fails. |
| 4 | Configure all slave boards to specify the secondary clock master as their clock fallback source. |

When the boards are configured in this way, the secondary clock master continues to drive the secondary clock (based on its own timing reference) if the primary clock master fails. Slave boards within the system fall back to synchronize their timing from the secondary clock master.

## Configuring clocking

You can configure board clocking in your system in one of the following ways. Choose only one of these configuration methods across all boards on the CT bus. Otherwise, the two methods interfere with one another, and board clocking will not operate properly.

| Method | Description |
|---|---|
| Using the *clockdemo* application model | Create an application that assigns each board its clocking mode, monitors clocking changes, and reconfigures clocking if clock fallback occurs.<br><br>A sample clocking application, *clockdemo*, is provided with Natural Access. *clockdemo* provides a robust fallback scheme that suits most system configurations. *clockdemo* source code is included, allowing you to modify the program if your clocking configuration is complex. For more information about *clockdemo*, refer to the *NMS OAM System User's Manual*.<br><br>Most clocking applications (including *clockdemo*) require that all boards on the CT bus be started in standalone mode. To learn how to set the board to start in standalone mode, refer to the board manual. |
| Using board keywords with or without application intervention | For each board on the CT bus, set the board keywords to determine the board's clocking mode and to determine how each board behaves if clock fallback occurs.<br><br>Unlike the *clockdemo* application, which allows several boards to take over mastery of the clock in a fallback situation, the board keyword method allows you to specify only a fixed primary and secondary master. For this reason, the board keyword method is best used only if you do not want to implement clock fallback in your system, or in test configurations where clock reliability is not a factor.<br><br>The board keyword method does not create an autonomous clock timing environment. An application must still intervene when clock fallback occurs to reset system clocking before other clocking changes occur. If both the primary and secondary clock masters stop driving the clocks (and an application does not intervene), the boards default to standalone mode. |

## Clock Management EMC

When a board is started or Hot Swap inserted, the Clock Management EMC configures the clock on the board as specified in the NMS OAM database. During system initialization, the Clock Management EMC also verifies that the bus clock master board (the board driving the clock) is running before any clock slave boards start up.

The Clock Management EMC exists as a managed object within NMS OAM and has keywords. For a list of these keywords, refer to *Clock Management EMC keyword summary* on page 141.

The Clock Management EMC provides H.100 and H.110 bus clock management services to boards in a chassis that are connected through the bus. As each board starts up, applications registered for NMS OAM alert notification can receive the following clocking-related events:

| Event | Description |
|---|---|
| CLKEVN_CONFIGURED | Board clock is successfully configured. |
| CLKEVN_INVALID_CONFIG_DATA | Board clock configuration is invalid or missing. |
| CLKEVN_CONFIG_FAILED | An attempt to configure board clock failed. |
| CLKEVN_OPEN_OAM_FAILED | An attempt to open a board plug-in failed. |
| CLKEVN_OPEN_SWITCH_FAILED | Could not open the board switch.<br>If the first switch open attempt fails, a second attempt is made. It is possible for an application to receive CLKEVN_CONFIGURED after CLKEVN_OPEN_SWITCH_FAILED. |

These events are received in the same manner as other NMS OAM events. For more information, see *Event buffer management* on page 29.

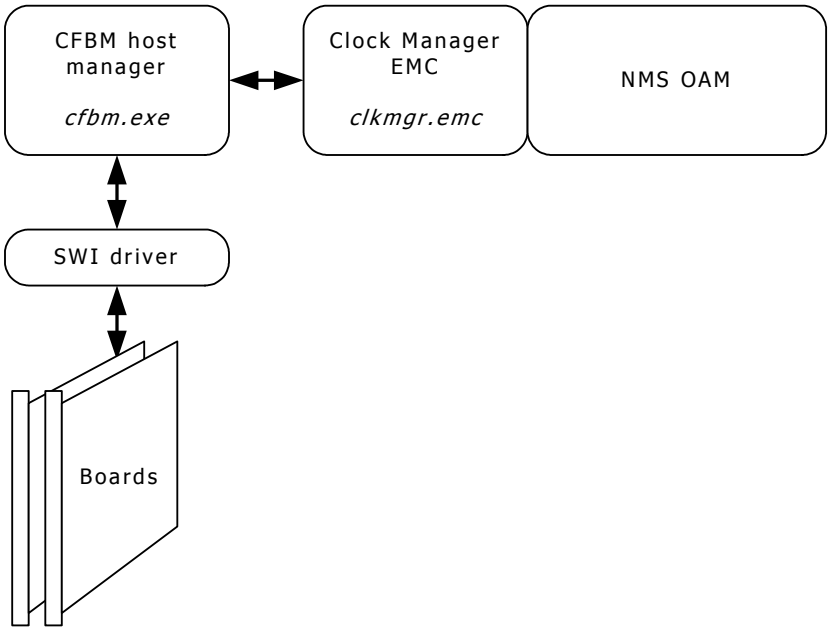
If clocking cannot be configured for a board, the board remains in standalone clock mode. In this mode, the board is not able to make connections to the CT bus.

### Board-level clock management architecture

The clock management component of NMS OAM consists of the following software modules:

| Module | Description |
|---|---|
| *clkmgr.emc* | The NMS OAM interface to clock management. This EMC has a managed object in the NMS OAM database, which can be accessed by an application. For more information, see *Clock Management EMC keyword summary* on page 141. |
| *cfbm.exe* | The clock manager service. *cfbm.exe* is a Natural Access application that is started by *clkmgr.emc*. It is started as a service under Windows. It is started as a daemon under UNIX.<br>To perform all of its clock management operations, *cfbm.exe* makes calls to the Natural Access Switching service (SWI). |

The following illustration shows the clock management architecture:



# 10 Function summary

## NMS OAM function overview

Natural Access provides synchronous and asynchronous functions.

| Function type | Description |
|---|---|
| Synchronous | Performs an operation and sends an indication to the application in the form of a return code. The return code is either SUCCESS or an error code. When a process invokes a synchronous function, no further processing takes place until the return code is received. Most OAM service functions are synchronous functions. |
| Asynchronous | Sends a return code to the application. However, this return code indicates only whether the function has been successfully initiated or not. If the return value is SUCCESS, actual execution results arrive later encapsulated in events. While waiting for the events, the application can proceed with other processing. |

The following table summarizes the differences between asynchronous and synchronous functions:

| Characteristic | Asynchronous | Synchronous |
|---|---|---|
| An operation is complete when a function returns. | No | Yes |
| An appropriate event returns when a function is complete. | Yes | No |
| A function can fail after a function returns. | Yes | No |

## Object configuration functions

Use the following functions to access managed objects and edit configuration data:

| Function | Synchronous/ Asynchronous | Description |
|---|---|---|
| **oamOpenObject** | Synchronous | Starts an editing session on the specified managed object. Returns a handle to be used with subsequent NMS OAM service object-specific function invocations. |
| **oamCloseObject** | Synchronous | Ends the editing session for a managed object, validating and saving changes. |
| **oamGetKeyword** | Synchronous | Retrieves a keyword value for a managed object. |
| **oamGetQualifier** | Synchronous | Retrieves a keyword qualifier for a managed object. |
| **oamSetKeyword** | Synchronous | Sets a keyword for a managed object. |
| **oamConfigExport** | Synchronous | Exports the current configuration to an output file. |
| **oamConfigImport** | Synchronous | Imports the current configuration from an input file. |

## Board information retrieval functions

Use the following functions to look up the name of a board:

| Function | Synchronous/ Asynchronous | Description |
|---|---|---|
| **oamBoardEnum** | Synchronous | Lists the names of all boards that exist as managed objects. |
| **oamBoardLookupByBusSlot** | Synchronous | Looks up a board by its PCI bus and slot and returns its name. |
| **oamBoardLookupByDriverIDs** | Synchronous | Looks up an AG board or a CG board by its driver ID information, and returns its name. |
| **oamBoardLookupByMACAddress** | Synchronous | Looks up a board by one of the MAC addresses of its Ethernet chips and returns its name. |
| **oamBoardLookupByNumber** | Synchronous | Looks up a board by its board number and returns its name. |
| **oamBoardLookupByProduct** | Synchronous | Given a product type, returns the board name of the first board of that product type listed in the NMS OAM database. |
| **oamBoardLookupBySerialNumber** | Synchronous | Looks up a board by its serial number and returns its name. |
| **oamBoardLookupByShelfSlot** | Synchronous | Looks up a board by its shelf and slot number and returns its name. CompactPCI CG boards in PICMG 2.1-compliant chassis only. |

Once the name of the board is determined, use the following functions to retrieve other types of information about the board:

| Function | Synchronous/ Asynchronous | Description |
|---|---|---|
| **oamBoardGetBusSlot** | Synchronous | Returns a board's PCI bus. |
| **oamBoardGetDriverIDs** | Synchronous | Returns an AG or CG board's driver ID information. |
| **oamBoardGetMACAddress** | Synchronous | Returns a board's primary MAC address. |
| **oamBoardGetNumber** | Synchronous | Returns a board's board number. |
| **oamBoardGetProduct** | Synchronous | Returns a board's product type. |
| **oamBoardGetSerialNumber** | Synchronous | Returns a board's serial number. |
| **oamBoardGetShelfSlot** | Synchronous | Returns a board's shelf and slot number. CompactPCI CG boards in PICMG 2.1-compliant chassis only. |
| **oamBoardGetStatus** | Synchronous | Returns a board's status. |

## Board management functions

Use the following functions to detect boards and create or delete managed objects:

| Function | Synchronous/ Asynchronous | Description |
|---|---|---|
| **oamCreateBoard** | Synchronous | Creates a new managed object for a board. |
| **oamDeleteBoard** | Asynchronous | Deletes a board managed object or all board managed objects. |
| **oamDetectBoards** | Synchronous | Detects physically installed boards in a resource host. AG and CG boards only. |
| **oamAddDetectedBoard** | Synchronous | Creates a managed object for a board previously detected by **oamDetectBoards**. Not currently supported. |

## Start, stop, and test board functions

Use the following functions to start, stop, and test boards:

| Function | Synchronous/ Asynchronous | Description |
|---|---|---|
| **oamStartBoard** | Asynchronous | Starts one or more boards. |
| **oamStopBoard** | Asynchronous | Stops one or more boards. |
| **oamTestBoard** | Asynchronous | Tests one or more CG boards. |

## NMS OAM event registration functions

Use the following functions to register or unregister:

| Function | Synchronous/ Asynchronous | Description |
|---|---|---|
| **oamAlertRegister** | Synchronous | Registers a client to receive events from NMS OAM. |
| **oamAlertUnregister** | Synchronous | Unregisters a client so it no longer receives events from NMS OAM. |

## Advanced resource host management functions

The NMS OAM service provides the following advanced resource host management functions (not intended for normal use):

| Function | Synchronous/ Asynchronous | Description |
|---|---|---|
| **oamShutdown** | Synchronous | Shuts down the Supervisor component of *ctdaemon*. |
| **oamSendBuffer** | Synchronous | Sends a raw data buffer to a CG board. This function is for debug purposes. |
| **oamAlertNotify** | Synchronous | Broadcasts a specified alert message to all clients registered for alert notification. |

# 11 Function reference

## Using the function reference

This section provides an alphabetical reference to the NMS OAM service functions. A prototype of each function is shown with the function description and details of all arguments and return values. A typical function includes:

| | |
|---|---|
| **Prototype** | The prototype is followed by a list of the function arguments. NMS data types include:<br>• WORD (16-bit unsigned)<br>• DWORD (32-bit unsigned)<br>• INT16 (16-bit signed)<br>• INT32 (32-bit signed)<br>• BYTE (8-bit unsigned)<br>If a function argument is a data structure, the complete data structure is defined.<br>[in] indicates that an argument is supplied to the function by the application.<br>[out] indicates that an argument is returned by the function. |
| **Return values** | The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated. Subsequent events indicate the status of the operation.<br>Refer to *NMS OAM error codes* on page 147 for a list of errors returned by the NMS OAM service functions. |
| **Events** | If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Refer to *NMS OAM events* on page 151 for a list of events returned by NMS OAM service functions.<br>Additional information such as reason codes and return values may be provided in the value field of the event. Refer to *NMS OAM reason codes* on page 155 for a list of reason codes returned with NMS OAM events. |
| **Details** | Additional information about the function. |
| **See also** | A list of related functions. |
| **Example** | Example functions that start with *my* are excerpts taken from sample application programs.<br>The notation `/* ... */` indicates additional code, which is not shown. |

## oamAddDetectedBoard

Adds a physically detected board to the active configuration. Not currently supported.

### Prototype

DWORD **oamAddDetectedBoard** ( CTAHD ***ctahd***, const char ***szName*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of a board to add. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | Function is not supported by this plug-in. |
| OAMERR_ALREADY_EXISTS | Board ***szName*** is already in the database, or one of the detected boards being added has the same PCI bus:slot as a board already in the database. |
| OAMERR_FILE_WRITE_ERROR | Error writing to database. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error. |
| OAMERR_NOT_FOUND | There are no boards to add. **oamDetectBoards** has not been called or has found no boards. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

Use **oamAddDetectedBoard** to create a managed object for one or more boards automatically detected by **oamDetectBoards**. Set ***szName*** to the name of the detected board to add. To add all detected boards at once, specify NULL as the board name.

When **oamAddDetectedBoard** is invoked, a new managed object is created on the resource host for the board. A record for the managed object is created in the NMS OAM database on the host containing default keyword settings.

**oamAddDetectedBoard** cannot add a detected board to the configuration if the board's name, PCI bus:slot location, or both matches that of another board already listed in the NMS OAM database. If **oamAddDetectedBoard** detects a conflict while attempting to add a single board, it returns OAMERR_ALREADY_EXISTS. If the conflict is detected while **oamAddDetectedBoard** is adding all boards (***szName*** is NULL), it returns OAMEVN_ALERT_INFO for each failed attempt. However, the function returns SUCCESS if at least one board has successfully been added to the database.

If you direct **oamAddDetectedBoard** to add all boards (***szName*** is NULL) and **oamDetectBoards** was not previously invoked or has found no boards, **oamAddDetectedBoard** returns OAMERR_NOT_FOUND.

## oamAlertNotify

Broadcasts a specified NMS OAM alert message to all client applications registered for alert notification.

### Prototype

DWORD **oamAlertNotify** ( CTAHD ***ctahd***, const OAM_MSG ****pMsg*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***pMsg*** | [in] Pointer to OAM_MSG structure containing the message to broadcast:<br>`typedef struct oam_msg_tag`<br>`{`<br>`  DWORD dwMsgLen;  // Message length, including  //`<br>`  // appended name and message strings//`<br>`  DWORD dwCode;  // Message code  //`<br>`  DWORD dwSeverity;  // Message severity  //`<br>`  DWORD dwOfsSzName;  // Offset to name string of  //`<br>`  // source managed object  //`<br>`  DWORD dwOfsSzMessage; // Offset to text message string  //`<br>`  // (String data is appended here) //`<br>`} OAM_MSG;` |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***pMsg*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_FILE_WRITE_ERROR | Error writing to database or file. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

Use **oamAlertNotify** to generate an alert notification that is received by all registered client applications on the same Natural Access Server. An application can use **oamAlertNotify** even if it is not registered for alert notification.

To generate a notification, the client application constructs an OAM_MSG structure for the notification and invokes **oamAlertNotify** to broadcast the message. With the function invocation, the application passes the context handle to the NMS OAM service and a pointer to the OAM_MSG structure. All registered client applications on the same Natural Access Server (including the sender) receive an event.

To receive events from the NMS OAM service and from installed EMCs (such as Hot Swap and Clock Management), a client application must register with the OAM service using **oamAlertRegister**.

## See also

**oamAlertUnregister**

## Example

```
DWORD sendAlert( CTAHD ctahd, const char *pMsg, const char *pName )
{
    DWORD    ret;
    char     msgBuf[ 256 ];
    OAM_MSG *pOamMsg;
    int      nSizeMsg, nSizeName;
    char    *pc;

    pOamMsg = (OAM_MSG *) msgBuf;
    pOamMsg->dwCode      = OAMEVN_ALERT;
    pOamMsg->dwSeverity  = CTA_TRACE_SEVERITY_INFO;

    /* point to text space after NMS OAM message information */
    pc = (char*) pOamMsg + sizeof(OAM_MSG);

    /* Copy object name and message into text buffer */
    nSizeName = strlen( pName ) + 1;
    nSizeMsg = strlen( pMsg ) + 1;

    strcpy( pc, pName );
    pc += nSizeName;
    strcpy( pc, pMsg );

    /* Set up offsets to various fields within the message */
    pOamMsg->dwOfsSzName = sizeof(OAM_MSG);
    pOamMsg->dwOfsSzMessage = sizeof(OAM_MSG) + nSizeName;
    pOamMsg->dwMsgLen = sizeof(OAM_MSG) + nSizeName + nSizeMsg;

    ret = oamAlertNotify( ctahd, pOamMsg );
    if ( ret != SUCCESS )
        printf( "Can't send alert message\n" );

    return ret;
}
```

## oamAlertRegister

Registers a client application to receive NMS OAM alert events.

### Prototype

DWORD **oamAlertRegister** ( CTAHD ***ctahd*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| OAMERR_ALREADY_EXISTS | Already registered. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error. |

### Details

If an application attempts to register a shared context using **oamAlertRegister**, and another application has already registered the context, OAMERR_ALREADY_EXISTS is returned.

For more information, see *Receiving NMS OAM alert notifications* on page 33.

### See also

**oamAlertNotify**, **oamAlertUnregister**

## oamAlertUnregister

Stops a client from receiving NMS OAM alert events.

### Prototype

DWORD **oamAlertUnregister** ( CTAHD ***ctahd*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error. |
| OAMERR_NOT_FOUND | Not currently registered. |

### Details

For more information, see *Receiving NMS OAM alert notifications* on page 33.

### See also

**oamAlertNotify**, **oamAlertRegister**

## oamBoardEnum

Lists the names of all boards in the NMS OAM database.

### Prototype

DWORD **oamBoardEnum** ( CTAHD ***ctahd***, int ***ixBoard***, char ****szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***ixBoard*** | [in] Board index. |
| ***szName*** | [out] Pointer to buffer containing board name. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return, including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified ***dwNameSize*** is insufficient. |
| OAMERR_NOT_FOUND | ***ixBoard*** is out of range. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamBoardEnum** lists the names of all boards that exist as managed objects. This list matches the list of boards in the Supervisor keyword Boards[x].

To use **oamBoardEnum**, pass an index value starting at 0 and continue until the function returns OAMERR_NOT_FOUND.

For more information, see *Retrieving board information* on page 47.

### Example

```
// List all the board names.
int const dwNameSize = 128;
char szName[ dwNameSize ];
for (int ixBoard=0; SUCCESS == oamBoardEnum( ctahd, ixBoard, szName,
                                             dwNameSize ); ++ixBoard)
     printf( "%s\n", szName );
```

## oamBoardGetBusSlot

Returns the PCI bus and slot number for a specified board. For more information, see *Retrieving board information* on page 47.

### Prototype

DWORD **oamBoardGetBusSlot** ( CTAHD ***ctahd***, const char ***szName***, DWORD ***pdwBus***, DWORD ***pdwSlot*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle |
| ***szName*** | [in] Pointer to a name of board to look up. |
| ***pdwBus*** | [out] Pointer to the board's PCI bus number. |
| ***pdwSlot*** | [out] Pointer to the board's slot number. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName***, ***pdwBus***, or ***pdwSlot*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### See also

**oamBoardGetDriverIDs**, **oamBoardGetMACAddress**, **oamBoardGetNumber**, **oamBoardGetProduct**, **oamBoardGetSerialNumber**, **oamBoardGetShelfSlot**, **oamBoardGetStatus**

## oamBoardGetDriverIDs

Returns the driver ID information for an AG or CG board with the given name.

### Prototype

DWORD **oamBoardGetDriverIDs** ( CTAHD ***ctahd***, const char ***szName***, char ***szDriverName***, DWORD ***dwDriverNameSize***, char ***szDriverBoardID***, DWORD ***dwDriverBoardIDSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to a name of board to look up. |
| ***szDriverName*** | [out] Pointer to buffer to receive driver name. |
| ***dwDriverNameSize*** | [in] Size of ***szDriverName*** (maximum number of bytes to return, including final null character). |
| ***szDriverBoardID*** | [out] Pointer to buffer to receive driver ID. |
| ***dwDriverBoardIDSize*** | [in] Size of ***dwDriverBoardID*** (maximum number of bytes to return, including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName***, ***szDriverName***, or ***szDriverBoardID*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

A board driver uses a driver ID to refer to a board. Typically (but not necessarily) the ID is an integer. The ID for a board is unique among all IDs associated with the driver. However, the ID is not necessarily unique among all boards in a resource host: two boards with different drivers can (by chance) have the same driver board ID number. Boards must be started when you invoke **oamBoardGetDriverIDs**.

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardGetBusSlot**, **oamBoardGetMACAddress**, **oamBoardGetNumber**, **oamBoardGetProduct**, **oamBoardGetSerialNumber**, **oamBoardGetShelfSlot**, **oamBoardGetStatus**

## oamBoardGetMACAddress

Returns a MAC address for the board with the given name.

### Prototype

DWORD **oamBoardGetMACAddress** ( CTAHD ***ctahd***, const char ***szName***, char ***szMACAddress***, DWORD ***dwMACAddressSize***, DWORD ***dwMACAddressIndex*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to look up. |
| ***szMACAddress*** | [out] Pointer to a set to the MAC address. |
| ***dwMACAddressSize*** | [in] Size of ***szMACAddress*** (maximum number of bytes to return, including final null character). |
| ***dwMACAddressIndex*** | [in] Set to the MAC address index number (between 1 and 4). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | An argument is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamBoardGetMACAddress** returns one of the MAC addresses of board ***szName***. MAC addresses are available only with boards with Ethernet interfaces. Each Ethernet interface on a board has a unique address.

Specify the index of the MAC address to return in ***dwMACAddressIndex***. If ***dwMACAddressIndex*** is not in the range of present MAC addresses on the board (usually 2), **oamBoardGetMACAddress** returns an empty string in ***szMACAddress***.

The returned ***szMACAddress*** is a string containing a standard MAC address; that is, 6 hexadecimal byte values separated by hyphens. For example:

```
E3-31-23-4A-2E-3F
```

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardGetBusSlot**, **oamBoardGetDriverIDs**, **oamBoardGetNumber**, **oamBoardGetProduct**, **oamBoardGetSerialNumber**, **oamBoardGetShelfSlot**, **oamBoardGetStatus**

## oamBoardGetNumber

Returns the board number for the board with the given name.

### Prototype

DWORD **oamBoardGetNumber** ( CTAHD ***ctahd***, const char ***szName***, DWORD ***pdwNumber*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to look up. |
| ***pdwNumber*** | [out] Set to the board number. |

### Return values

| Return values | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** or ***pdwNumber*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

A board number identifies a board in a resource host. The number is unique among all boards in a host. Many Natural Access services use the board number to identify a board.

A default board number is assigned to a board when a managed object for the board is created using **oamCreateBoard**. Change the board number by setting the Number keyword for the board's managed object.

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardGetBusSlot**, **oamBoardGetDriverIDs**, **oamBoardGetMACAddress**, **oamBoardGetProduct**, **oamBoardGetSerialNumber**, **oamBoardGetShelfSlot**, **oamBoardGetStatus**

## oamBoardGetProduct

Returns the product type for the board with the given name.

### Prototype

DWORD **oamBoardGetProduct** ( CTAHD ***ctahd***, const char ***szName***, char ***szProduct***, DWORD ***dwProductSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to look up. |
| ***szProduct*** | [out] Pointer to buffer to receive board's product type. |
| ***dwProductSize*** | [in] Size of ***szProduct*** (maximum number of bytes to return, including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** or ***szProduct*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | NMS OAM supervisor is shut down. |

### Details

The product type returned by **oamBoardGetProduct** is one of the product types listed in the Supervisor keyword Products[x].

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardGetBusSlot**, **oamBoardGetDriverIDs**, **oamBoardGetMACAddress**, **oamBoardGetNumber**, **oamBoardGetSerialNumber**, **oamBoardGetShelfSlot**, **oamBoardGetStatus**

## oamBoardGetSerialNumber

Returns the serial number for the board with the given name.

### Prototype

DWORD **oamBoardGetSerialNumber** ( CTAHD ***ctahd***, const char ***szName***, char ***szSerialNumber***, DWORD ***dwSerialNumberSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to look up. |
| ***szSerialNumber*** | [out] Pointer to buffer to receive serial number of board. |
| ***dwSerialNumberSize*** | [in] Size of ***szSerialNumber*** (maximum number of bytes to return, including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** or ***szSerialNumber*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

Boards must be started when you invoke **oamBoardGetSerialNumber**. For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardGetBusSlot**, **oamBoardGetDriverIDs**, **oamBoardGetMACAddress**, **oamBoardGetNumber**, **oamBoardGetProduct**, **oamBoardGetShelfSlot**, **oamBoardGetStatus**

## oamBoardGetShelfSlot

Returns the shelf and slot for the board with the given name. This information is available only for CompactPCI CG boards in PICMG 2.1-compliant chassis.

### Prototype

DWORD **oamBoardGetShelfSlot** ( CTAHD ***ctahd***, const char ***szName***, DWORD ***pdwShelf***, DWORD ***pdwSlot*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to look up. |
| ***pdwShelf*** | [out] Pointer to the shelf number of the chassis containing the board. |
| ***pdwSlot*** | [out] Pointer to the board's slot number. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName***, ***pdwShelf***, or ***pdwSlot*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamBoardGetShelfSlot** returns the shelf and slot values of board ***szName***.Shelf. Shelf and slot information is available only for CompactPCI CG boards in a PICMG 2.1-compliant chassis.

Shelf and slot values are coded in 5 bits each, so ***pdwShelf*** and ***pdwSlot*** are both values between 0 and 31. If shelf and slot values cannot be retrieved (for example, if board ***szName*** is a PCI board), ***pdwShelf*** and ***pdwSlot*** are set to illegal value 32.

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardGetBusSlot**, **oamBoardGetDriverIDs**, **oamBoardGetMACAddress**, **oamBoardGetNumber**, **oamBoardGetProduct**, **oamBoardGetSerialNumber**, **oamBoardGetStatus**

## oamBoardGetStatus

Queries the board status.

### Prototype

DWORD **oamBoardGetStatus** ( CTAHD ***ctahd***, const char ***szName***, INT32 ***dataDescriptor***, INT32 ***data** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to look up. |
| ***dataDescriptor*** | [in] Specifies the query data.<br>The following table lists the valid values for ***dataDescriptor*** and the boards that each value supports: |
| ***data*** | [out] Pointer to the query result. For status types, see the ***dataDescriptor*** argument. |

| Value | Description | Supported CG boards |
|---|---|---|
| 0 | CPU utilization during the last second of operation, as a percentage of available CPU. | All |
| 1 | Average CPU utilization over the last sixteen seconds of operation, as a percentage of available CPU. | All |
| 2 | Heap usage, in bytes. | All |
| 3 | Available memory, in bytes. | All |
| 4 | Main temperature sensor, in tenths of a degree Celsius. | All |
| 5 | CPU temperature sensor, in tenths of a degree Celsius. | CG 6565/c/e |
| 6 | DSP temperature sensor, in tenths of a degree Celsius. | CG 6565/c/e |
| 7 | IO temperature sensor, in tenths of a degree Celsius. | CG 6565/c/e |
| 8 | Fan tachometer, in RPM. | CG 6565/e |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName***, ***pdwShelf***, or ***pdwSlot*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_NOT_SUPPORTED_WARNING | Board is not a CG board, or the requested data is not available on the board. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

The application must specify the type of query data. Available data depends on the board type, as described in the description of ***dataDescriptor***.

### See also

**oamBoardGetBusSlot**, **oamBoardGetDriverIDs**, **oamBoardGetMACAddress**, **oamBoardGetNumber**, **oamBoardGetProduct**, **oamBoardGetSerialNumber**, **oamBoardGetShelfSlot**

## oamBoardLookupByBusSlot

Looks up a board by its PCI bus and slot and returns its name. For more information, see *Retrieving board information* on page 47.

### Prototype

DWORD **oamBoardLookupByBusSlot** ( CTAHD ***ctahd***, DWORD ***dwBus***, DWORD ***dwSlot***, char ***szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***dwBus*** | [in] PCI bus number of board to look up. |
| ***dwSlot*** | [in] Number of board to look up. |
| ***szName*** | [out] Pointer to buffer to receive name of board. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### See also

**oamBoardLookupByDriverIDs**, **oamBoardLookupByMACAddress**, **oamBoardLookupByNumber**, **oamBoardLookupByProduct**, **oamBoardLookupByShelfSlot**

## oamBoardLookupByDriverIDs

Looks up an AG or CG board by its driver ID information and returns its name.

### Prototype

DWORD **oamBoardLookupByDriverIDs** ( CTAHD ***ctahd***, const char ***szDriverName***, const char ***szDriverBoardID***, char ***szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szDriverName*** | [in] Pointer to a driver name for which to search. |
| ***szDriverBoardID*** | [in] Pointer to a driver board ID for which to search. |
| ***szName*** | [out] Pointer to buffer to receive name of board. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName***, ***szDriverName***, or ***szDriverBoardID*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

A driver board ID is used by a given board driver to refer to a board. Typically (but not necessarily) the ID is an integer. The ID for a board is unique among all IDs associated with the driver. However, the ID is not necessarily unique among all boards in a host: two boards with different drivers can (by chance) have the same driver board ID number. Boards must be started when you invoke this function.

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardLookupByBusSlot**, **oamBoardLookupByMACAddress**, **oamBoardLookupByNumber**, **oamBoardLookupByProduct**, **oamBoardLookupByShelfSlot**

## oamBoardLookupByMACAddress

Looks up a board by one of the MAC addresses of its Ethernet interfaces and returns its name.

### Prototype

DWORD **oamBoardLookupByMACAddress** ( CTAHD ***ctahd***, const char ***szMACAddress***, char ***szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szMACAddress*** | [in] Pointer to a MAC address of one of the Ethernet chips on the board to look up. |
| ***szName*** | [out] Pointer to buffer to receive name of board. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamBoardLookupByMACAddress** retrieves the name of the board with MAC address ***szMACAddress***. MAC addresses are available only with boards with Ethernet interfaces. Each chip has a unique address.

***szMACAddress*** is a string containing a standard MAC address; that is, 6 hexadecimal byte values separated by hyphens. For example:

```
E3-31-23-4A-2E-3F
```

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardLookupByBusSlot**, **oamBoardLookupByDriverIDs**, **oamBoardLookupByProduct**, **oamBoardLookupByShelfSlot**, **oamBoardLookupByShelfSlot**

## oamBoardLookupByNumber

Looks up a board by its board number and returns its name.

### Prototype

DWORD **oamBoardLookupByNumber** ( CTAHD ***ctahd***, DWORD ***dwNumber***, char ***szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***dwNumber*** | [in] Number of board to look up. |
| ***szName*** | [out] Pointer to buffer to receive name of board. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

A board number is a number used to identify a board in a resource host. The number is unique among all boards in a host. Many Natural Access services use the board number to identify a board.

A default board number is assigned to a board when a managed object for the board is created using **oamCreateBoard**. To change the board number, set the Number keyword for the board managed object.

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardLookupByBusSlot**, **oamBoardLookupByDriverIDs**, **oamBoardLookupByMACAddress**, **oamBoardLookupByProduct**

## oamBoardLookupByProduct

Given a product type, returns the board name of the first board of that product type listed in the NMS OAM database.

### Prototype

DWORD **oamBoardLookupByProduct** ( CTAHD ***ctahd***, const char ***szProduct*** char ***szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szProduct*** | [in] Pointer to a product type to look up. |
| ***szName*** | [out] Pointer to buffer to receive name of board. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** or ***szProduct*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

The specified product type must be one of the product types listed in the Supervisor keyword Products[x]. Only the board name of the first board of the specified product type is returned. For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardLookupByBusSlot**, **oamBoardLookupByDriverIDs**, **oamBoardLookupByMACAddress**, **oamBoardLookupByNumber**, **oamBoardLookupByShelfSlot**

## oamBoardLookupBySerialNumber

Looks up a board by its serial number and returns its name.

### Prototype

DWORD **oamBoardLookupBySerialNumber** ( CTAHD ***ctahd***, const char ***szSerialNumber***, char ***szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szSerialNumber*** | [in] Pointer to a serial number to look up. |
| ***szName*** | [out] Pointer to buffer to receive name of board. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** or ***szSerialNumber*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

Boards must be booted when you invoke **oamBoardLookupBySerialNumber**. For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardLookupByBusSlot**, **oamBoardLookupByDriverIDs**, **oamBoardLookupByMACAddress**, **oamBoardLookupByNumber**, **oamBoardLookupByProduct**, **oamBoardLookupByShelfSlot**

## oamBoardLookupByShelfSlot

Looks up a board by its shelf and slot number and returns its name. This information is available only for CompactPCI CG boards in PICMG 2.1-compliant chassis.

### Prototype

DWORD **oamBoardLookupByShelfSlot** ( CTAHD ***ctahd***, DWORD ***dwShelf***, DWORD ***dwSlot***, char ***szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***dwShelf*** | [in] Shelf number of chassis containing board to look up. |
| ***dwSlot*** | [in] Slot number of board to look up. |
| ***szName*** | [out] Pointer to buffer to receive name of board. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | Board not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamBoardLookupByShelfSlot** returns the name of the board with shelf and slot values ***dwShelf*** and ***dwShelf***. Shelf and slot information is available only for CompactPCI CG boards in PICMG 2.1-compliant chassis.

***dwShelf*** and ***dwShelf*** must be in the range 0 - 31 (5 bits).

For more information, see *Retrieving board information* on page 47.

### See also

**oamBoardLookupByBusSlot**, **oamBoardLookupByDriverIDs**, **oamBoardLookupByMACAddress**, **oamBoardLookupByNumber**, **oamBoardLookupByProduct**, **oamBoardLookupBySerialNumber**

## oamCloseObject

Ends the editing session for a managed object, validating and saving changes.

### Prototype

DWORD **oamCloseObject** ( HMOBJ ***hObject*** )

| Argument | Description |
|---|---|
| ***hObject*** | [in] Handle of managed object to close (returned by **oamOpenObject**). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_INVALID_HANDLE | ***hObject*** is invalid. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | Object is not open. |
| OAMERR_ALREADY_EXISTS | Attempt to rename or renumber to a value already in use. |
| OAMERR_FILE_WRITE_ERROR | Error writing to database. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error. |
| OAMERR_NOT_FOUND | Object not found (***hObject*** invalid). |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamCloseObject** closes a managed object previously opened by **oamOpenObject**.

In addition to the return values previously listed, this function can return keyword validation errors specific to the type of managed object being closed. For more information about these errors, refer to the documentation for the managed component.

For more information about opening and closing objects, see *Accessing configuration data* on page 35.

### Example

```
DWORD closeObject( CTAHD ctahd, HMOBJ objHd )
{
    DWORD ret;

    /* Close a handle to a previously opened managed object */
    if ( ( ret = oamCloseObject( objHd ) ) != SUCCESS )
    {
        char errMsg[ 100 ];

        ctaGetText( ctahd, ret, errMsg, sizeof(errMsg) );
        printf("oamCloseObject failed: %s\n", errMsg );
    }

    return ret;
}
```

## oamConfigExport

Exports the current configuration to an output file.

### Prototype

DWORD **oamConfigExport** ( CTAHD ***ctahd***, const char ***szFile*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szFile*** | [in] Pointer to a file name for the configuration data output file. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szFile*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | Managed objects are open for writing. |
| OAMERR_FILE_READ_ERROR | Error reading from database. |
| OAMERR_FILE_WRITE_ERROR | Error writing to file. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamConfigExport** exports the entire contents of the NMS OAM database to a file. All managed objects currently open for writing must be closed before using this function. Otherwise, any pending changes will not be included in the exported data.

The resulting output file is for use only with **oamConfigImport** and must not be edited in any way.

For more information, see *Importing and exporting configuration data* on page 43.

## oamConfigImport

Imports the current configuration from an input file.

### Prototype

DWORD **oamConfigImport** ( CTAHD ***ctahd***, const char ***szFile*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szFile*** | [in] Pointer to a file name for the configuration data input file. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szFile*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | Managed objects are open. |
| OAMERR_FILE_READ_ERROR | Error reading from file. |
| OAMERR_FILE_WRITE_ERROR | Error writing to database. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamConfigImport** imports the entire configuration from a specified input file created by **oamConfigExport**. The new configuration replaces the contents of the NMS OAM database.

**Note:** Close all managed objects before invoking this function.

For more information, refer to *Importing and exporting configuration data* on page 43.

## oamCreateBoard

Creates a new managed object for a board.

### Prototype

DWORD **oamCreateBoard** ( CTAHD ***ctahd***, const char ***szProduct***, char ***szName***, DWORD ***dwNameSize*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szProduct*** | [in] Pointer to a product type. |
| ***szName*** | [out] Pointer to buffer to receive object name. |
| ***dwNameSize*** | [in] Size of ***szName*** (maximum number of bytes to return including the final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** or ***szProduct*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_FILE_WRITE_ERROR | Error writing to database. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error or the specified buffer size is too small. |
| OAMERR_NOT_FOUND | ***szProduct*** not found (is invalid). |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamCreateBoard** creates a new managed object for a board of product type ***szProduct***. Each board plug-in supports a different set of product types. The products supported by the set of plug-ins installed on a resource host can be determined by examining the Supervisor keyword Products[x].

If the function succeeds, a new managed object is created. A record containing default keyword settings is created for the object in the NMS OAM database. A default name and board number are generated for the board.

The board name is passed back in the function call. The name is added to the list of board names in the Supervisor keyword Boards[x].

Retrieve the board number using any **oamBoardLookupBy*XXX*** function, as described in *Retrieving board information* on page 47.

To change the board name and board number, modify the Name or Number keywords for the board managed object. Refer to *Accessing configuration data* on page 35.

For more information about creating managed objects, see *Managing board information* on page 48.

### See also

**oamDeleteBoard**

## oamDeleteBoard

Deletes the managed object for a board or all board managed objects.

### Prototype

DWORD **oamDeleteBoard** ( CTAHD ***ctahd***, const char ***szName*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to delete. Specify NULL to delete all board managed objects. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | Managed object is opened by another application. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error or the specified buffer size is too small. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

**Note:** Other board-specific errors can also be received. For information about possible board errors, see the board-specific documentation.

### Events

| Event | Description |
|---|---|
| OAMEVN_DELETED | Board object was deleted. |
| OAMEVN_STOPBOARD_DONE | Function completed successfully. The event value field contains the result of the board starting attempt:<br>OAM_REASON_FINISHED<br>Board stopped successfully.<br>OAM_REASON_FAILED<br>Board stopping attempt failed. Further information is available in the OAM_MSG message buffer. |

### Details

**oamDeleteBoard** deletes the managed object for a specified board or for all boards. The specified boards must exist as managed objects. To obtain the names of all managed boards, examine the Supervisor keyword Boards[x].

**oamDeleteBoard** is an asynchronous function. If the board was running when **oamDeleteBoard** was invoked, it is stopped before the managed object is deleted.

An application cannot delete a board managed object if it is currently open by another application.

For more information about deleting managed objects, see *Managing board information* on page 48.

### See also

**oamCreateBoard**

## oamDetectBoards

Physically detects AG and CG boards.

### Prototype

DWORD **oamDetectBoards** ( CTAHD ***ctahd*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | Function is not supported by this plug-in. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

**Note:** Other board-specific errors can also be received.

### Details

**oamDetectBoards** searches the chassis for boards. When it finds a board, it adds an entry to the DetectedBoards[] array in the Supervisor managed object, and stores information about the board in subkeywords as shown in the following table:

| This subkeyword... | Contains... |
|---|---|
| DetectedBoards[x].Name | Generated board name |
| DetectedBoards[x].Product | Board product type |
| DetectedBoards[x].Location.PCI.Bus | PCI bus location |
| DetectedBoards[x].Location.PCI.Slot | PCI slot location |

The DetectedBoards[] array is empty until this function is called. You can retrieve the value of the DetectedBoards.Count keyword to indicate the number of boards in the DetectedBoards[] array.

For more information, refer to *Detecting installed boards automatically* on page 49.

### See also

**oamAddDetectedBoard**

## oamGetKeyword

Retrieves a keyword value for the managed object with the specified handle.

### Prototype

DWORD **oamGetKeyword** ( HMOBJ ***hObject***, const char ***szKeyword***, const char ***szValue***, DWORD ***dwValueSize*** )

| Argument | Description |
|---|---|
| ***hObject*** | [in] Managed object to get keyword name and value from (returned by **oamOpenObject**). |
| ***szKeyword*** | [in] Pointer to the name of keyword. |
| ***szValue*** | [out] Pointer to buffer containing value assigned to ***szKeyword***. |
| ***dwValueSize*** | [in] Size of ***szValue*** (maximum number of bytes to return, including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szValue*** or ***szKeyword*** is NULL or 0. |
| CTAERR_INVALID_HANDLE | ***hObject*** is invalid. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | ***hObject*** is not open. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size is too small. |
| OAMERR_NOT_FOUND | ***szKeyword*** not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamGetKeyword** retrieves a keyword value for the managed object with the specified handle (returned by **oamOpenObject**). The object must be open. For more information about keywords, see *Accessing configuration data* on page 35.

### See also

**oamGetQualifier**, **oamSetKeyword**

## Example

```
DWORD getKeyword( CTAHD ctahd, HMOBJ objHd, char *pKeyword, char *value )
{
    DWORD ret;

    /* Get the value of the passed in keyword. The managed object
    *  has previously been opened and a handle to it (objHd) has been
    *  provided.
    */
    ret = oamGetKeyword( objHd, pKeyword, value, sizeof(value) );

    if ( ret != SUCCESS )
    {
        char errMsg[ 100 ];

        ctaGetText( ctahd, ret, errMsg, sizeof(errMsg) );
        printf("oamGetKeyword failed: %s\n", errMsg );
    }

    return ret;
}
```

## oamGetQualifier

Retrieves a keyword qualifier for a keyword in the specified managed object.

### Prototype

DWORD **oamGetQualifier** ( HMOBJ ***hObject***, const char ***szKeyword***, const char ***szQualifier***, char ***szValue***, DWORD ***dwValueSize*** )

| Argument | Description |
|---|---|
| ***hObject*** | [in] Managed object to get keyword qualifier from (returned by **oamOpenObject**). |
| ***szKeyword*** | [in] Pointer to the name of keyword. |
| ***szQualifier*** | [in] Pointer to the name of the qualifier for the keyword. |
| ***szValue*** | [out] Pointer to a buffer to receive keyword qualifier value. |
| ***dwValueSize*** | [in] Size of ***szValue*** (maximum number of bytes to return, including final null character). |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szKeyword***, ***szQualifier***, or ***szValue*** is NULL or 0. |
| CTAERR_INVALID_HANDLE | ***hObject*** is invalid. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | ***hObject*** is not open. |
| OAMERR_NOT_ENOUGH_MEMORY | Specified buffer size ***dwValueSize*** is too small. |
| OAMERR_NOT_FOUND | ***szKeyword*** or ***szQualifier*** not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamGetQualifier** retrieves the keyword qualifier for a specified keyword for the managed object with the specified handle (returned by **oamOpenObject**). The object must be open.

For more information about keywords and qualifiers, see *Accessing configuration data* on page 35.

### See also

**oamGetKeyword**, **oamSetKeyword**

## Example

```
DWORD getType( CTAHD ctahd, HMOBJ objHd, char *pKeyword, char *qualText )
{
    DWORD ret;

    /* Get the "Type" attribute of the passed in keyword. The managed
    *  object has previously been opened and a handle to it (objHd)
    *  is being provided.
    */
    ret = oamGetQualifier( objHd, pKeyword, "Type",
                           qualText, sizeof(qualText) );

    if ( ret != SUCCESS )
    {
        char errMsg[ 100 ];

        ctaGetText( ctahd, ret, errMsg, sizeof(errMsg) );
        printf("oamGetQualifier failed: %s\n", errMsg );
    }

    return ret;
}
```

## oamOpenObject

Starts an editing session on the specified managed object. Returns a handle to be used with subsequent NMS OAM service function invocations.

### Prototype

DWORD **oamOpenObject** ( CTAHD ***ctahd***, const char ****szName***, HMOBJ ****phObject***, DWORD ***dwMode*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to a name of the managed object. |
| ***phObject*** | [out] Pointer to the handle of the managed object. |
| ***dwMode*** | [in] Specifies open mode. See the Details section for valid values. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** or ***phObject*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | ***phObject*** is open to another application, and open modes do not allow use. |
| OAMERR_FILE_READ_ERROR | Error reading from database. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error. |
| OAMERR_NOT_FOUND | ***phObject*** not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

To access configuration data in a managed object, a client application opens the object using **oamOpenObject**. With this function invocation, the application supplies an opening mode. Valid modes are:

| Mode | Description |
|---|---|
| OAM_READONLY | Opens the object for reading only. The application cannot change keyword values. |
| OAM_READWRITE | Opens the object for both reading and writing. The application can change the values of read and write keywords.<br>A managed object cannot be opened in this mode if any other client application has it currently open. |

If a client application attempts to open a managed object for writing, and the object is currently open by another application, **oamOpenObject** fails and OAMERR_ACCESS_DENIED is returned to the application.

**Note:** A managed object can be opened in read-only mode (OAM_READONLY) even if it is currently open in read-only mode by another application.

If the object-opening attempt succeeds, **oamOpenObject** returns a handle to the object, to be used with subsequent editing functions.

For more information about opening and closing objects, see *Accessing configuration data* on page 35.

### See also

**oamCloseObject**

### Example

```
DWORD openObject( CTAHD ctahd, char *objName, HMOBJ *pObjHd, int bDoWrite )
{
    DWORD ret;
    DWORD mode = OAM_READONLY;
    DWORD retry = 0;

    if ( bDoWrite )

        /* Caller wants to performs writes to keywords within the
        *  managed object.
        */
        mode = OAM_READWRITE;

    else

        /* Only reads will be performed on the keywords
        *  within the managed object.
        */
        mode = OAM_READONLY;

    do
    {
        if ( retry )
            /* Another thread or process has the object opened for write */
            Sleep( 10 );

        ret = oamOpenObject( ctahd, objName, pObjHd, mode );

    } while ( ( ret == OAMERR_ACCESS_DENIED ) && (retry++ < 5);

    if ( ret != SUCCESS )
    {
        char errMsg[ 100 ];

        ctaGetText( ctahd, ret, errMsg, sizeof(errMsg) );
        printf("oamOpenObject failed: %s\n", errMsg );
    }

    return ret;
}
```

## oamSendBuffer

Sends a raw buffer to a CG board.

### Prototype

DWORD **oamSendBuffer** ( CTAHD ***ctahd***, const char ***szName***, const BYTE ***pbBuf***, DWORD ***dwBufLen*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to which to send the buffer. |
| ***pbBuf*** | [in] Pointer to buffer to send. |
| ***dwBufLen*** | [in] Size of ***pbBuf***. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szName*** or ***pbBuf*** is NULL or 0. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_NOT_FOUND | Board ***szName*** not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Details

**oamSendBuffer** is an advanced function. Sending data using this method can place the board in an indeterminate state, or can cause the resource host to malfunction in other ways. This functionality is provided mainly for plug-in developers, to permit easy low-level debugging. To use this function, contact NMS technical services.

CG boards must be started before you invoke **oamSendBuffer**.

To send a raw buffer of data to a board, construct the buffer and invoke **oamSendBuffer**, passing the name of the target board and a pointer to the data buffer in the function invocation. In addition to the return values previously listed, this function can return board type-specific values. For information about possible board errors, see the board-specific documentation.

## oamSetKeyword

Sets a keyword for a managed object.

### Prototype

DWORD **oamSetKeyword** ( HMOBJ ***hObject***, const char ***szKeyword***, const char ***szValue*** )

| Argument | Description |
|---|---|
| ***hObject*** | [in] Handle to the managed object. |
| ***szKeyword*** | [in] Pointer to the name of keyword. |
| ***szValue*** | [in] Pointer to a buffer containing keyword value to set. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_BAD_ARGUMENT | ***szValue*** or ***szKeyword*** is NULL or 0. |
| CTAERR_INVALID_HANDLE | ***hObject*** is invalid. |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | ***hObject*** is not open for writing. |
| OAMERR_NOT_ENOUGH_MEMORY | Memory allocation error. |
| OAMERR_NOT_FOUND | ***szKeyword*** not found. |
| OAMERR_SERVER_NOT_RUNNING | ***szKeyword*** not found. |

**Note:** Other object-specific keyword validation errors can also occur. For more information about these errors, refer to the documentation for the managed component.

### Details

If the managed object was opened in OAM_READWRITE mode, **oamSetKeyword** sets the value of a keyword. **oamSetKeyword** does not create the keyword if it does not exist; otherwise it returns an error. For more information about modes, see *oamOpenObject* on page 104.

### See also

**oamGetKeyword**, **oamGetQualifier**

## Example

```
DWORD setKeyword( CTAHD ctahd, HMOBJ objHd, char *pKeyword, char *value )
{
    DWORD ret;

    /* Set the value of the passed in keyword. The managed object
    *  has previously been opened in OAM_READWRITE mode and a handle
    *  to it (objHd) has been provided.
    */
    ret = oamSetKeyword( objHd, pKeyword, value );

    if ( ret != SUCCESS )
    {
        char errMsg[ 100 ];

        ctaGetText( ctahd, ret, errMsg, sizeof(errMsg) );
        printf("oamSetKeyword failed: %s\n", errMsg );
    }

    return ret;
}
```

## oamShutdown

Shuts down the Supervisor.

### Prototype

DWORD **oamShutdown** ( CTAHD ***ctahd*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |

### Return values

SUCCESS

### Details

To use **oamShutdown**, contact NMS technical services.

The client must end the session after this call with no further calls to the host.

## oamStartBoard

Starts one or more boards.

### Prototype

DWORD **oamStartBoard** ( CTAHD ***ctahd***, const char ***szName*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to start. Set to NULL to start all boards in parallel. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_CANT_CREATE_THREAD | Cannot create thread to do asynchronous operations. |
| OAMERR_NOT_FOUND | Board ***szName*** not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Events

| Event | Description |
|---|---|
| OAMEVN_STARTBOARD_DONE | Function completed successfully. The event value field contains the result of the board starting attempt:<br>OAM_REASON_FINISHED<br>Board started successfully.<br>OAM_REASON_FAILED<br>Board start attempt failed. Further information is available in the OAM_MSG message buffer. |

### Details

**oamStartBoard** starts the specified board, or all boards in parallel. Because this function can incur delay, it is an asynchronous call that returns immediately and is followed by an alert notification on completion.

For more information, see *Starting, stopping, and testing boards* on page 50.

### See also

**oamStopBoard**, **oamTestBoard**

## oamStopBoard

Stops one or more boards.

### Prototype

DWORD **oamStopBoard** ( CTAHD ***ctahd***, const char ***szName** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Name of board to stop. Set to NULL to stop all boards. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_CANT_CREATE_THREAD | Cannot create thread to do asynchronous operations. |
| OAMERR_NOT_FOUND | Board ***szName*** not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Events

| Event | Description |
|---|---|
| OAMEVN_STOPBOARD_DONE | Function completed successfully. The event value field contains the result of the attempt to stop the board:<br>OAM_REASON_FINISHED<br>Board stopped successfully.<br>OAM_REASON_FAILED<br>Board stop attempt failed. Further information is available in the OAM_MSG message buffer. |
| OAMEVN_STOPBOARD_REQ | Request was made to stop a board. The board will be stopped in 150 ms. |

**Note:** In addition to these return values, this function can return board type-specific values. For more information about possible board errors, see the board-specific documentation.

### Details

**oamStopBoard** stops the specified board, or all boards in parallel. Because this function can incur delay, it is an asynchronous call that returns immediately and is followed by an alert notification on completion.

**Note: oamStopBoard** stops boards regardless of any tasks they are currently performing. To avoid problems, make sure that a board is idle before stopping it.

For more information, see *Starting, stopping, and testing boards* on page 50.

### See also

**oamStartBoard**, **oamTestBoard**

## oamTestBoard

Tests one or more CG boards.

### Prototype

DWORD **oamTestBoard** ( CTAHD ***ctahd***, const char ***szName***, DWORD ***dwOptions*** )

| Argument | Description |
|---|---|
| ***ctahd*** | [in] Context handle. |
| ***szName*** | [in] Pointer to the name of board to test. Set to NULL to test all boards in parallel. |
| ***dwOptions*** | [in] Test level bit mask. |

### Return values

| Return value | Description |
|---|---|
| SUCCESS | |
| CTAERR_SVR_COMM | Natural Access Server is not running. |
| OAMERR_ACCESS_DENIED | Function is not supported by this plug-in. |
| OAMERR_CANT_CREATE_THREAD | Cannot create thread to do asynchronous operations. |
| OAMERR_NOT_FOUND | Board ***szName*** not found. |
| OAMERR_SERVER_NOT_RUNNING | Supervisor is shut down. |

### Events

| Event | Description |
|---|---|
| OAMEVN_TESTBOARD_DONE | Function completed successfully. The event value field contains the result of the board testing attempt:<br>OAM_REASON_FINISHED<br>Board tested successfully.<br>OAM_REASON_FAILED<br>Board test attempt failed. Further information is available in the OAM_MSG message buffer. |

### Details

**oamTestBoard** tests the specified CG boards, or all boards in parallel. Because this function can incur delay, it is an asynchronous call that returns immediately and is followed by an alert notification on completion. For more information, see *Starting, stopping, and testing boards* on page 50. CG boards must be stopped when you invoke **oamTestBoard**.

### See also

**oamStartBoard**, **oamStopBoard**

# 12 Demonstration program

## Keyword enumeration and setting: oaminfo

*oaminfo* displays and sets keyword information for NMS OAM managed objects. Use *oaminfo* to perform the following tasks:

- Retrieve keyword information
- Search for keyword names
- Set keyword information
- Work with a remote server

### Usage

```
oaminfo [options]
```

where ***options*** are:

| Option | Description |
|---|---|
| -n ***name*** | Specifies the name of managed object. |
| -b ***boardno*** | Specifies the board number. Can be used instead of -n option. Default = 0. |
| -k ***keyword*** | Specifies the keyword name. If unspecified, shows all keywords. |
| -f ***field*** | Specifies the desired field within keyword name ***keyword***. |
| -i ***index*** | Specifies the array index of keyword. Default = all. |
| -v ***value*** | Sets value of keyword to ***value***. |
| -q | Shows keyword qualifiers. |
| -s ***text*** | Displays all keyword names containing specified ***text***. |
| -h | Displays usage information and terminate. |
| -@ ***host*** | Performs specified operations on a resource host ***host***. If the host is not specified, the operations are performed on the host where the utility was initialized. |

### Featured functions

**oamBoardLookupByNumber**, **oamCloseObject**, **oamGetKeyword**, **oamGetQualifier**, **oamOpenObject**, **oamSetKeyword**

### Description

This demonstration program performs keyword enumeration on one keyword in a managed object, or on all keywords in a managed object if no keyword is specified. It can also search for text in keywords and set keyword values.

## Retrieving keyword information

To retrieve a list of all keyword names and settings in a managed object, specify the object with the -n option, as shown in this example:

```
oaminfo -n "Supervisor"
```

*oaminfo* returns information similar to the following:

```
ExtendedManagementComponents[0] = HotSwap.emc
ExtendedManagementComponents[1] = clkmgr.emc
Products[0] = CG_6000C_QUAD
Products[1] = QX 2000/100-4L
Products[2] = QX 2000/200-4L
Products[3] = QX 2000/80-1L
Products[4] = QX 2000/80-4L
BoardPlugins[0] = cg6kpi.bpi
BoardPlugins[1] = qx2kpi.bpi
Boards[0] = MyBoard 0
Boards[1] = MyBoard 1
Name = Supervisor
AutoStartEnabled = No
AutoStopEnabled = No
```

For board managed objects, specify the managed object either by its name with the -n option or by its board number with the -b option:

```
oaminfo -b 0
```

The object name for the NMS OAM Supervisor is Supervisor. The object name for a plug-in or EMC is its file name (for example, *hotswap.emc*).

To retrieve the setting of a particular keyword name in a managed object, specify the managed object, and then the keyword with the -k option:

```
oaminfo -n "Supervisor" -k AutoStartEnabled
```

*oaminfo* returns information similar to the following:

```
AutoStartEnabled = No
```

To retrieve EEPROM keyword information, use the -k option and specify the full path of the requested keyword. For example:

```
oaminfo -k Eeprom.NumTrunk
```

To retrieve the settings of all keywords in an array, specify the array keyword name with the -k option:

```
oaminfo -n "Supervisor" -k Boards
```

*oaminfo* returns information similar to the following:

```
Boards[0] = MyBoard 0
Boards[1] = MyBoard 1
```

To retrieve the settings of a specific keyword within an array, also specify the keyword's index with the -i option:

```
oaminfo -n "Supervisor" -k Boards -i 1
```

*oaminfo* returns information similar to the following:

```
Boards[1] = MyBoard 1
```

To retrieve the qualifiers for the keywords you specify, use the -q option:

```
oaminfo -n "Supervisor" -k Boards -q
```

*oaminfo* returns information similar to the following:

```
Boards[0] = MyBoard 0
STRING Type
Read-only
Description: Board name

Boards[1] = MyBoard 1
STRING Type
Read-only
Description: Board name
```

### Searching for keyword names

*oaminfo* can perform simple searches for keyword names containing certain characters. To do so, specify the characters using the -s option, as shown:

```
oaminfo -n "Supervisor" -s Auto
```

*oaminfo* returns information similar to the following:

```
AutoStartEnabled = Yes
AutoStopEnabled = Yes
```

You can combine the search function with other functions. For example:

```
oaminfo -n "Supervisor" -s Auto -q
```

This command line returns:

```
AutoStartEnabled = No
        STRING Type
        Read-Write
        Description: enables automatic start of boards
        There are 2 possible values:
                Yes
                No

AutoStopEnabled = No
        STRING Type
        Read-Write
        Description: enables automatic stop of boards
        There are 2 possible values:
                Yes
                 No
```

### Setting keyword information

To set the value of a read and write keyword, specify the keyword using the -k option and the value using the -v option, as shown:

```
oaminfo -n "Supervisor" -k AutoStart -v Yes
```

*oaminfo* returns information similar to the following:

```
Currently AutoStartEnabled = No
Now AutoStartEnabled = Yes
```

### Working with a remote server

To display or set keyword values for an NMS OAM managed object on a remote host, use the -@ command line option followed by the host name. For example, the following command displays a list of all keyword names and settings in the Supervisor managed object on remote host MyHost1:

```
oaminfo -n "Supervisor" -@ MyHost1
```

# 13 NMS OAM Supervisor keyword reference

## Supervisor keyword summary

The following table summarizes the Supervisor keywords that you can change. The Readonly qualifier for these keywords is set to No:

| To specify whether the board is... | Use this keyword... |
|---|---|
| Started automatically when the Supervisor starts or when the board is Hot Swap inserted | AutoStartEnabled |
| Stopped automatically when the Supervisor stops | AutoStopEnabled |

The following table summarizes the keywords that cannot be changed. The Readonly qualifier for these keywords is set to Yes:

| To... | Use these keywords... |
|---|---|
| Determine what software components are installed | BoardPlugins[x]<br>ExtendedManagementComponents[x] |
| Retrieve a list of board names that are in the configuration database | Boards[x] |
| Retrieve a list of boards detected using **oamDetectBoards** | DetectedBoards[x].Location.PCI.Bus<br>DetectedBoards[x].Location.PCI.Slot<br>DetectedBoards[x].Name<br>DetectedBoards[x].Product |
| Retrieve information about the Supervisor managed object | Name<br>Version.Major<br>Version.Minor |
| Retrieve a list of all products supported by the set of plug-ins currently installed | Products[x] |

For more information, refer to the following topics:

| To learn more about... | Refer to... |
|---|---|
| Setting and retrieving keywords | *Accessing configuration data* on page 35 |
| Specifying keywords in configuration files using NMS OAM utilities | *NMS OAM System User's Manual* |

## Using the keyword reference

The keywords are presented in detail in the following topics. Each keyword description includes:

| | |
|---|---|
| **Syntax** | The syntax of the keyword |
| **Access** | Indicates if the keyword's value can be retrieved and changed:<br>Read/Write - Value can be retrieved or changed.<br>Read-only - Value can be retrieved only.<br>Write-only - Value can be changed only. Retrieved value contains no information. |
| **Type** | The data type of the value: String or Integer |
| **Default** | Default value |
| **Allowed values** | A list of all possible values |
| **Example** | An example of usage |
| **Details** | A detailed description of the keyword's function |
| **See also** | A list of related keywords |

## AutoStartEnabled

If set to Yes, the Supervisor attempts to start all boards whenever it starts up or when a board is Hot Swap-inserted.

### Syntax

AutoStartEnabled = ***flag***

### Access

Read/Write

### Type

String

### Default

No

### Allowed values

Yes | No

### Example

```
AutoStartEnabled = No
```

### Details

The Supervisor starts a board automatically only if the AutoStart keyword in the board's managed object is set to Yes.

For more information about starting boards automatically, refer to the *NMS OAM System User's Manual*.

### See also

AutoStopEnabled

## AutoStopEnabled

If set to Yes, the Supervisor attempts to stop all boards whenever it shuts down.

### Syntax

AutoStopEnabled = ***flag***

### Access

Read/Write

### Type

String

### Default

No

### Allowed values

Yes | No

### Example

```
AutoStopEnabled = No
```

### Details

The Supervisor stops a board automatically only if the AutoStop keyword in the board's managed object is set to Yes.

For more information about stopping boards automatically, refer to the *NMS OAM System User's Manual*.

### See also

AutoStartEnabled

## BoardPlugins[x]

Contains a list of the names of all currently installed board plug-ins.

### Syntax

BoardPlugins[***x***] = ***plug_in_name***

### Access

Read-only

### Type

Array

### Details

When the Supervisor starts up, it loads all board plug-ins and EMCs that it finds. The Supervisor adds the name of each plug-in to the BoardPlugins[***x***] array keyword. The name of a given plug-in is its file name. This name matches the value of the Name keyword in the plug-in's managed object.

To determine the number of names in BoardPlugins[***x***], retrieve the value of BoardPlugins.Count.

The Supervisor looks for plug-ins and EMCs in the *nms\bin* directory (*/opt/nms/lib* under UNIX). Plug-in files have the *.bpi* extension. EMC files have the *emc* extension.

### See also

ExtendedManagementComponents[x]

## Boards[x]

Contains a list of names of all boards managed by NMS OAM (the list of all boards that have managed objects in the NMS OAM database). **oamBoardEnum** returns the same list of boards.

### Syntax

Boards[***x***] = ***boardname***

### Access

Read-only

### Type

Array

### Details

To determine the number of names in Boards[***x***], retrieve the value of Boards.Count.

## DetectedBoards[x].Location.PCI.Bus

Contains the PCI bus location of a board detected by **oamDetectBoards**.

### Syntax

DetectedBoards[***x***].Location.PCI.Bus = ***detectedboardbusloc***

### Access

Read-only

### Type

Integer

### Details

This keyword and all other DetectedBoards[***x***].***xxx*** keywords are assigned values when **oamDetectBoards** is invoked. NMS OAM searches the chassis for boards. When it finds a board, it adds an array entry to the DetectedBoards[] array and stores information about the board in subkeywords of the new array entry. The DetectedBoards[] array is renewed whenever the function is called. It is empty until this function is called.

When the board search is complete, the application can create managed objects for the boards in the array. To learn how to create managed objects for detected boards, refer to *Detecting installed boards automatically* on page 49.

**Note:** Only AG and CG boards can be automatically detected.

You can retrieve the value of the DetectedBoards.Count keyword to indicate the number of boards in the DetectedBoards[] array.

When board detection is complete, the DetectedBoards[] array includes boards that already have managed objects in the NMS OAM database.

### See also

DetectedBoards[x].Location.PCI.Slot, DetectedBoards[x].Name, DetectedBoards[x].Product

## DetectedBoards[x].Location.PCI.Slot

Contains the PCI slot location of a board detected by **oamDetectBoards**.

### Syntax

DetectedBoards[***x***].Location.PCI.Slot = ***detectedboardslotloc***

### Access

Read-only

### Type

Integer

### Details

This keyword and all other DetectedBoards[***x***].***xxx*** keywords are assigned values when **oamDetectBoards** is invoked. NMS OAM searches the chassis for boards. When it finds a board, it adds an array entry to the DetectedBoards[] array and stores information about the board in subkeywords of the new array entry. The DetectedBoards[] array is renewed whenever the function is called. It is empty until this function is called.

When the board search is complete, the application can create managed objects for the boards in the array. To learn how to create managed objects for detected boards, refer to *Detecting installed boards automatically* on page 49.

**Note:** Only AG and CG boards can be automatically detected.

You can retrieve the value of the DetectedBoards.Count keyword to indicate the number of boards in the DetectedBoards[] array.

When board detection is complete, the DetectedBoards[] array includes boards that already have managed objects in the NMS OAM database.

### See also

DetectedBoards[x].Location.PCI.Bus, DetectedBoards[x].Name, DetectedBoards[x].Product

## DetectedBoards[x].Name

Contains a unique temporary board name for a board detected by **oamDetectBoards**.

### Syntax

DetectedBoards[***x***].Name = ***detectedboardname***

### Access

Read-only

### Type

String

### Details

The board name is automatically generated during board detection. The name can be changed later, after a managed object is created for the board.

This keyword and all other DetectedBoards[***x***].***xxx*** keywords are assigned values when **oamDetectBoards** is invoked. NMS OAM searches the chassis for boards. When it finds a board, it adds an array entry to the DetectedBoards[] array and stores information about the board in subkeywords of the new array entry. The DetectedBoards[] array is renewed whenever the function is called. It is empty until this function is called.

When the board search is complete, the application can create managed objects for the boards in the array. To learn how to create managed objects for detected boards, refer to *Detecting installed boards automatically* on page 49.

**Note:** Only AG and CG boards can be automatically detected.

You can retrieve the value of the DetectedBoards.Count keyword to indicate the number of boards in the DetectedBoards[] array.

When board detection is complete, the DetectedBoards[] array includes boards that already have managed objects in the NMS OAM database.

### See also

DetectedBoards[x].Location.PCI.Bus, DetectedBoards[x].Location.PCI.Slot, DetectedBoards[x].Product

## DetectedBoards[x].Product

Contains the product type of a board detected by **oamDetectBoards**.

### Syntax

DetectedBoards[***x***].Product = ***detectedboardproducttype***

### Access

Read-only

### Type

String

### Details

This keyword and all other DetectedBoards[***x***].***xxx*** keywords are assigned values when **oamDetectBoards** is invoked. NMS OAM searches the chassis for boards. When it finds a board, it adds an array entry to the DetectedBoards[] array and stores information about the board in subkeywords of the new array entry. The DetectedBoards[] array is renewed whenever the function is called. It is empty until this function is called.

When the board search is complete, the application can create managed objects for the boards in the array. To learn how to create managed objects for detected boards, refer to *Detecting installed boards automatically* on page 49.

**Note:** Only AG and CG boards can be automatically detected.

You can retrieve the value of the DetectedBoards.Count keyword to indicate the number of boards in the DetectedBoards[] array.

When board detection is complete, the DetectedBoards[] array includes boards that already have managed objects in the NMS OAM database.

### See also

DetectedBoards[x].Location.PCI.Bus, DetectedBoards[x].Location.PCI.Slot, DetectedBoards[x].Name

## ExtendedManagementComponents[x]

Contains a list of all currently installed EMCs.

### Syntax

ExtendedManagementComponents[***x***] = ***EMCname***

### Access

Read-only

### Type

Array

### Details

When the Supervisor starts up, it loads all board plug-ins and EMCs that it finds. The Supervisor adds the name of each EMC to the ExtendedManagementComponents[***x***] array keyword. The name of a given EMC is its file name. This name matches the value of the Name keyword in the EMC's managed object.

The number of names in ExtendedManagementComponents[***x***] can be determined by retrieving the value of ExtendedManagementComponents.Count.

The Supervisor looks for plug-ins and EMCs in the *\nms\bin* directory (*/opt/nms/lib* under UNIX). Plug-in files have the *.bpi* extension. EMC files have the *.emc* extension.

### See also

BoardPlugins[x]

## Name

Contains the name of the Supervisor managed object.

### Syntax

Name = ***name***

### Access

Read-only

### Type

String

### Details

Use this name when referencing the Supervisor managed object in software.
Expected value: Supervisor.

## Products[x]

Contains a list of names of supported products.

### Syntax

Products[***x***] = ***productname***

### Access

Read-only

### Type

Array

### Details

When the Supervisor initializes, it queries all plug-ins for the names of products they support. Each of these product types is stored in this list.

The number of names in Products[***x***] can be determined by retrieving the value of Products.Count.

## Version.Major

Contains the software version number of the Supervisor (major half).

**Syntax**

Version.Major = ***majorversionno***

**Access**

Read-only

**Type**

Integer

**See also**

Version.Minor

## Version.Minor

Contains the software version number of the Supervisor (minor half).

**Syntax**

Version.Minor = ***minorversionno***

**Access**

Read-only

**Type**

Integer

**See also**

Version.Major

# 14 Hot Swap EMC keyword reference

## Hot Swap EMC keyword summary

The following table summarizes the Hot Swap EMC keywords that you can change. The Readonly qualifier for these keywords is set to No:

| To... | Use this keyword... |
|---|---|
| Initiate insertion or extraction of a board | Board.boardname.Command |

The following table summarizes the keywords that cannot be changed. The Readonly qualifier for these keywords is set to Yes:

| To... | Use these keywords... |
|---|---|
| Determine the Hot Swap state of a board | Board.boardname.State |
| Retrieve information about the Hot Swap managed object | Name<br>Version.Major<br>Version.Minor |

For more information, refer to the following topics:

| To learn more about... | Refer to... |
|---|---|
| Using Hot Swap | *Hot Swap board insertion and removal processes* on page 53 |
| Setting and retrieving keywords | *Accessing configuration data* on page 35 |
| Specifying keywords in configuration files using NMS OAM utilities | *NMS OAM System User's Manual* |
| Setting and retrieving keywords from the command line | *Keyword enumeration and setting: oaminfo* on page 113 |

## Using the Hot Swap EMC keyword reference

The keywords are presented in detail in the following topics. Each keyword description includes:

| | |
|---|---|
| **Syntax** | The syntax of the keyword |
| **Access** | Indicates if the keyword's value can be retrieved and changed:<br>Read/Write - Value can be retrieved or changed.<br>Read-only - Value can be retrieved only.<br>Write-only - Value can be changed only. Retrieved value contains no information. |
| **Type** | The data type of the value: String or Integer |
| **Default** | Default value |
| **Allowed values** | A list of all possible values |
| **Example** | An example of usage |
| **Details** | A detailed description of the keyword's function |
| **See also** | A list of related keywords |

## Board.boardname.Command

Initiates insertion or extraction of a board. These actions correspond to the user actions of opening and closing the board ejector handles. To initiate extraction, set the keyword to Extract. To initiate insertion, set the keyword to Insert.

### Syntax

Board.***boardname***.Command = ***command***

### Access

Write-only

### Type

String

### Allowed values

| Return value | Description |
|---|---|
| ***boardname*** | Any board name currently in the NMS OAM database. |
| ***command*** | Either Insert or Extract |

### Example

```
Board.MyBoard.Command = Insert
```

### Details

This keyword takes effect as soon as it is set.

Board.boardname.Command is a write-only keyword. You cannot query it to learn if a board is inserted or extracted. To determine the Hot Swap state of a board, use Board.boardname.State.

## Board.boardname.State

Returns the current Hot Swap state of the board.

### Syntax

Board.***boardname***.State = ***state***

### Access

Read-only

### Type

String

### Details

The following table shows expected values for this keyword:

| Argument | Expected values |
|---|---|
| ***boardname*** | Any board name string. |
| ***state*** | Any of the following state names:<br>• Extracted<br>• OffLine<br>• OnLinePending<br>• Failed<br>• OnLine<br>• OffLinePending<br>• Unsupported |

For more information, refer to *Hot Swap state machine* on page 56.

## Name

Contains the name of the Hot Swap managed object.

### Syntax

Name = ***name***

### Access

Read-only

### Type

String

### Details

Use this name when referencing the Hot Swap managed object in software. Expected value: *hotswap.emc*.

## Version.Major

Contains the software version number of the Hot Swap EMC (major half).

### Syntax

Version.Major = ***majorversionno***

### Access

Read-only

### Type

Integer

### See also

Version.Minor

## Version.Minor

Contains the software version number of the Hot Swap EMC (minor half).

**Syntax**

Version.Minor = ***minorversionno***

**Access**

Read-only

**Type**

Integer

**See also**

Version.Major

# 15 Clock Management EMC keyword reference

## Clock Management EMC keyword summary

The following table summarizes the Clock Management EMC keywords that you can change. The Readonly qualifier for these keywords is set to No:

| To... | Use this keyword... |
|---|---|
| Apply the clock configuration to all boards in the resource host | Apply |

The following table summarizes the keywords that cannot be changed. The Readonly qualifier for these keywords is set to Yes:

| To... | Use these keywords... |
|---|---|
| Retrieve information about the Clock Management managed object | Name<br>Version.Major<br>Version.Minor |

For more information, refer to the following topics:

| To learn more about... | Refer to... |
|---|---|
| Configuring clocks | *H.100/H.110 clocking overview* on page 59 |
| Setting and retrieving keywords | *Accessing configuration data* on page 35 |
| Specifying keywords in configuration files using NMS OAM utilities | *NMS OAM System User's Manual* |

## Using the Clock Management EMC keyword reference

The keywords are presented in detail in the following topics. Each keyword description includes:

| | |
|---|---|
| **Syntax** | The syntax of the keyword |
| **Access** | Indicates if the keyword's value can be retrieved and changed:<br>Read/Write - Value can be retrieved or changed.<br>Read-only - Value can be retrieved only.<br>Write-only - Value can be changed only. Retrieved value contains no information. |
| **Type** | The data type of the value: String or Integer |
| **Default** | Default value |
| **Allowed values** | A list of all possible values |
| **Example** | An example of usage |
| **Details** | A detailed description of the keyword's function |
| **See also** | A list of related keywords |

## Apply

When set to Yes, the Clock Management EMC applies all clock management keyword settings to each board in the resource host.

### Syntax

Apply = ***command***

### Access

Read/Write

### Type

String

### Default

No

### Allowed values

Yes | No

### Example

```
Apply = Yes
```

### Details

Clock configuration settings normally go into effect when boards are started. You can use the Apply keyword to cause the settings to take place immediately without restarting the boards.

The Apply keyword takes effect as soon as it is set to Yes. The EMC immediately sets the keyword to No again after the clock management settings are applied. Thus if you query the value of the Apply keyword (with *oaminfo* or by other means) it always returns No.

This keyword is for testing purposes only. Data on the CT bus can be disrupted when this keyword is set.

For more information about clock configuration settings, refer to *H.100/H.110 clocking overview* on page 59.

## Name

Contains the name of the Clock Management managed object.

### Syntax

Name = ***name***

### Access

Read-only

### Type

String

### Description

Use this name when referencing the Clock Management managed object in software. Expected value: *clkmgr.emc*.

## Version.Major

Contains the software version number of the Clock Management EMC (major half).

**Syntax**

Version.Major = ***majorversionno***

**Access**

Read-only

**Type**

Integer

**See also**

Version.Minor

## Version.Minor

Contains the software version number of the Clock Management EMC (minor half).

**Syntax**

Version.Minor = ***minorversionno***

**Access**

Read-only

**Type**

Integer

**See also**

Version.Major

# 16 Errors, events, and reasons

## NMS OAM error codes

All NMS OAM service functions return a status code. If the return code is not SUCCESS (0), it is an error code indicating that the function has failed and a reason for the failure.

Error codes can also appear in the value field of a DONE event. Use the CTA_IS_ERROR macro to determine if a value is an error.

Errors are returned by the NMS OAM Supervisor and by any installed EMC. The prefix of the error indicates its source:

| Error prefix | Source | Defined in |
|---|---|---|
| OAMERR_ | NMS OAM service | *oamdef.h* |
| HSWERR_ | Hot Swap EMC | *hswdef.h* |
| CLKERR_ | Clock Management EMC | *clkdef.h* |
| CTAERR_ | Natural Access | *ctadef.h* |
| (Other) | (Other service) | (Other .h file) |

This topic provides a summary of NMS OAM error codes. The error codes are presented in two tables:

- Alphabetical, by error name, including a description of the problem and a possible solution.
- Numerical, by hexadecimal value, decimal value, and error name.

### Alphabetical error summary

The following table provides an alphabetical list of errors. All errors are 32 bits.

| Error name | Hex | Decimal | Description |
|---|---|---|---|
| CLKERR_FAILURE | 0x40010002 | 1073807362 | A general failure occurred. |
| CLKERR_NO_RESOURCES | 0x40010000 | 1073807360 | No resources are available. |
| CLKERR_NOT_FOUND | 0x40010001 | 1073807361 | Specified object is not found. |
| HSWERR_FAILURE | 0x40110001 | 1074855937 | A general failure occurred. |
| HSWERR_INVALID_BUS | 0x40110003 | 1074855939 | Specified PCI bus is invalid. |
| HSWERR_NO_RESOURCES | 0x40110002 | 1074855938 | No resources are available. |
| HSWERR_NOT_FOUND | 0x40110004 | 1074855940 | Specified object is not found. |
| OAMERR_ACCESS_DENIED | 0x40000006 | 1073741830 | The operation cannot be completed, because it is not allowed. |

| Error name | Hex | Decimal | Description |
|---|---|---|---|
| OAMERR_ALREADY_EXISTS | 0x40000005 | 1073741829 | The operation cannot be completed because an item exists that should not already exist. |
| OAMERR_BOARD_SELFTEST_FAIL | 0x4000010E | 1073742094 | An attempt to test a board failed. |
| OAMERR_BOARD_SELFTEST_INVALID | 0x4000010F | 1073742095 | Board testing is not supported by this plug-in. |
| OAMERR_CANT_CREATE_THREAD | 0x40000009 | 1073741833 | Cannot create thread to do asynchronous operation. |
| OAMERR_CLOSE_OBJECT_FAIL | 0x4000010B | 1073742091 | The Supervisor could not close a managed object in the NMS OAM database. |
| OAMERR_CREATE_BOARD_FAIL | 0x40000105 | 1073742085 | The Supervisor could not create a managed object for a board in the NMS OAM database. |
| OAMERR_DESTORY_BOARD_FAIL (sic) | 0x40000106 | 1073742086 | The Supervisor could not delete a managed object for a board from the NMS OAM database. |
| OAMERR_FILE_READ_ERROR | 0x40000007 | 1073741831 | Error reading from database or file. |
| OAMERR_FILE_WRITE_ERROR | 0x40000008 | 1073741832 | Error writing to database or file. |
| OAMERR_GET_BOARD_KEYWORD_FAIL | 0x40000107 | 1073742087 | The Supervisor could not retrieve a keyword value from the NMS OAM database. |
| OAMERR_GET_QUALIFIER_FAIL | 0x40000109 | 1073742089 | The Supervisor could not retrieve a qualifier for a keyword from the NMS OAM database. |
| OAMERR_NOT_ENOUGH_MEMORY | 0x40000003 | 1073741827 | Memory allocation error, or specified buffer size is too small. |
| OAMERR_NOT_FOUND | 0x40000004 | 1073741828 | Specified board, object, keyword, or qualifier is not found. |
| OAMERR_NOT_SUPPORTED_WARNING | 0x40000103 | 1073742083 | An operation is not supported. |
| OAMERR_OPEN_DRIVER_FAIL | 0x40000111 | 1073742097 | An attempt to open a driver failed. |
| OAMERR_OPEN_OBJECT_FAIL | 0x4000010A | 1073742090 | The Supervisor could not open a managed object in the NMS OAM database. |
| OAMERR_PLUGIN_EXIT_FAIL | 0x40000102 | 1073742082 | The Supervisor could not shut down a plug-in. |
| OAMERR_PLUGIN_INIT_FAIL | 0x40000101 | 1073742081 | The Supervisor could not initialize a plug-in. |
| OAMERR_SEND_BUFFER_FAIL | 0x40000110 | 1073742096 | An attempt to send a raw buffer to a board with **oamSendBuffer** failed. |

| Error name | Hex | Decimal | Description |
|---|---|---|---|
| OAMERR_SERVER_NOT_FOUND | 0x40000001 | 1073741825 | Natural Access Server is not found. |
| OAMERR_SERVER_NOT_RUNNING | 0x40000002 | 1073741826 | NMS OAM Supervisor is shut down. |
| OAMERR_SET_BOARD_KEYWORD_FAIL | 0x40000108 | 1073742088 | The Supervisor could not set a keyword value in the NMS OAM database. |
| OAMERR_START_BOARD_FAIL | 0x4000010C | 1073742092 | An attempt to start a board failed. |
| OAMERR_STOP_BOARD_FAIL | 0x4000010D | 1073742093 | An attempt to stop a board failed. |

## Numerical error summary

The following table provides a list of errors in numerical order:

| Hex | Decimal | Error name |
|---|---|---|
| 0x40000001 | 1073741825 | OAMERR_SERVER_NOT_FOUND |
| 0x40000002 | 1073741826 | OAMERR_SERVER_NOT_RUNNING |
| 0x40000003 | 1073741827 | OAMERR_NOT_ENOUGH_MEMORY |
| 0x40000004 | 1073741828 | OAMERR_NOT_FOUND |
| 0x40000005 | 1073741829 | OAMERR_ALREADY_EXISTS |
| 0x40000006 | 1073741830 | OAMERR_ACCESS_DENIED |
| 0x40000007 | 1073741831 | OAMERR_FILE_READ_ERROR |
| 0x40000008 | 1073741832 | OAMERR_FILE_WRITE_ERROR |
| 0x40000009 | 1073741833 | OAMERR_CANT_CREATE_THREAD |
| 0x40000101 | 1073742081 | OAMERR_PLUGIN_INIT_FAIL |
| 0x40000102 | 1073742082 | OAMERR_PLUGIN_EXIT_FAIL |
| 0x40000103 | 1073742083 | OAMERR_NOT_SUPPORTED_WARNING |
| 0x40000105 | 1073742085 | OAMERR_CREATE_BOARD_FAIL |
| 0x40000106 | 1073742086 | OAMERR_DESTORY_BOARD_FAIL (sic) |
| 0x40000107 | 1073742087 | OAMERR_GET_BOARD_KEYWORD_FAIL |
| 0x40000108 | 1073742088 | OAMERR_SET_BOARD_KEYWORD_FAIL |
| 0x40000109 | 1073742089 | OAMERR_GET_QUALIFIER_FAIL |
| 0x4000010A | 1073742090 | OAMERR_OPEN_OBJECT_FAIL |
| 0x4000010B | 1073742091 | OAMERR_CLOSE_OBJECT_FAIL |
| 0x4000010C | 1073742092 | OAMERR_START_BOARD_FAIL |
| 0x4000010D | 1073742093 | OAMERR_STOP_BOARD_FAIL |

| Hex | Decimal | Error name |
|---|---|---|
| 0x4000010E | 1073742094 | OAMERR_BOARD_SELFTEST_FAIL |
| 0x4000010F | 1073742095 | OAMERR_BOARD_SELFTEST_INVALID |
| 0x40000110 | 1073742096 | OAMERR_SEND_BUFFER_FAIL |
| 0x40000111 | 1073742097 | OAMERR_OPEN_DRIVER_FAIL |
| 0x40010000 | 1073807360 | CLKERR_NO_RESOURCES |
| 0x40010001 | 1073807361 | CLKERR_NOT_FOUND |
| 0x40010002 | 1073807362 | CLKERR_FAILURE |
| 0x40110001 | 1074855937 | HSWERR_FAILURE |
| 0x40110002 | 1074855938 | HSWERR_NO_RESOURCES |
| 0x40110003 | 1074855939 | HSWERR_INVALID_BUS |
| 0x40110004 | 1074855940 | HSWERR_NOT_FOUND |

## NMS OAM events

Each event's prefix relates the event to a specific NMS Communications library of functions, as described in the following table:

| Event prefix | Source | Defined in |
|---|---|---|
| OAMEVN_ | NMS OAM service | *oamdef.h* |
| HSWEVN_ | Hot Swap EMC | *hswdef.h* |
| CLKEVN_ | Clock Management EMC | *clkdef.h* |
| CTAEVN_ | Natural Access | *ctadef.h* |
| (Other) | (Other service) | (Other .h file) |

To receive events from the NMS OAM service and from installed EMCs (such as Hot Swap and Clock Management), a client application must register for alerts with the NMS OAM service. To learn how to register an application, see *Receiving NMS OAM alert notifications* on page 33.

The following table alphabetically lists the NMS OAM service events:

| Event name | Hex | Decimal | Description |
|---|---|---|---|
| CLKEVN_CONFIG_FAILED | 0x40012003 | 1073815555 | An attempt to configure board's clock failed. |
| CLKEVN_CONFIGURED | 0x40012001 | 1073815553 | Board clock is successfully configured. |
| CLKEVN_INVALID_CONFIG_DATA | 0x40012002 | 1073815554 | Board clock configuration is invalid or missing. |
| CLKEVN_OPEN_OAM_FAILED | 0x40012004 | 1073815556 | An attempt to open a board plug-in failed. |
| CLKEVN_OPEN_SWITCH_FAILED | 0x40012005 | 1073815557 | Could not open board's switch. |
| HSWEVN_BOARD_INSERTED | 0x40112005 | 1074864133 | A board was physically inserted. No board resources are yet available for the application. The state changes from Extracted to OffLine, and the Hot Swap driver starts to configure the board on the PCI bus.<br>If the board was successfully configured on the PCI bus, the application receives HSWEVN_ONLINE_PENDING and the state changes to OnLinePending. Otherwise, the application receives HSWEVN_PCI_CONFIG_FAILED and the state stays OffLine. |
| HSWEVN_BOARD_OFFLINE | 0x40112003 | 1074864131 | The board is in the OffLine state and can be extracted. If the board was removed from the OffLinePending state without waiting for HSWEVN_BOARD_OFFLINE (surprise extraction), then HSWEVN_BOARD_REMOVED immediately follows. |

| Event name | Hex | Decimal | Description |
|---|---|---|---|
| HSWEVN_BOARD_READY | 0x40112001 | 1074864129 | An inserted board was successfully configured and booted by the OAM service. The state changes to OnLine. After receiving this event, an application can use the inserted board. All resources are accessible.<br>**Note:** Board clocking configuration may not yet be complete. If the board is operating in a clocking mode other than Standalone, the application must wait for CLKEVN_CONFIGURED before using the board. |
| HSWEVN_BOARD_REMOVED | 0x40112004 | 1074864132 | A board was physically removed from the chassis. The Hot Swap state changes from OffLine to Extracted. |
| HSWEVN_ONLINE_PENDING | 0x40112006 | 1074864134 | The Hot Swap manager created a device instance for the board and notified the NMS OAM service to start it. The state changes to OnLinePending.<br>If the device instance is successfully created and the board is started, the application receives HSWEVN_BOARD_READY and the state changes to OnLine. Otherwise, it receives HSWEVN_PREPARATION_FAILED and the state changes to Failed.<br>If extraction was initiated by *hsmon* or with the Board.***boardname***.Command keyword, and the board handles are open in the Failed state, the application receives HSWEVN_BOARD_OFFLINE and the state changes to OffLine.<br>If insertion was initiated in the Failed state by *hsmon* or with the Board.***boardname***.Command keyword, the application receives HSWEVN_ONLINE_PENDING and the process repeats as previously described. |
| HSWEVN_PCI_CONFIG_FAILED | 0x40112007 | 1074864135 | The board transitioned to the OffLine state and the PCI configuration process failed.<br>There are three ways to make another configuration attempt:<br>• Open and close the handles on the board.<br>• Initiate insertion from *hsmon*.<br>• Initiate insertion using the Board.***boardname***.Command keyword. |

| Event name | Hex | Decimal | Description |
|---|---|---|---|
| HSWEVN_PREPARATION_FAILED | 0x40112008 | 1074864136 | The application receives this event in the OnLinePending state when the device instance was not successfully created or the board failed to start. The state changes to Failed. There are three ways to try to start the board:<br>• Open and close the handles on the board.<br>• Initiate insertion from *hsmon*.<br>• Initiate insertion using the Board.***boardname***.Command keyword.<br>If extraction was initiated by *hsmon* or with the Board.***boardname***.Command keyword, and the board handles are open in the Failed state, the application receives HSWEVN_BOARD_OFFLINE and the state changes to OffLine.<br>If insertion was initiated in the Failed state by *hsmon* or with the Board.***boardname***.Command keyword, the application receives HSWEVN_ONLINE_PENDING and the preparation process repeats.<br>A board fails to start with this event if the Supervisor keyword AutoStartEnabled or the board keyword AutoStart is set to No. |
| HSWEVN_REMOVAL_REQUESTED | 0x40112002 | 1074864130 | A board's ejector handles were opened or extraction was initiated from *hsmon*. This event warns the application that the board will be removed. After receiving this event, an application must close all resources associated with the board. The state changes to OffLinePending. When an application closes all resources, it receives HSWEVN_BOARD_OFFLINE and the state changes to OffLine. |
| OAMEVN_ALERT | 0x40002001 | 1073750017 | Alert indication. The OAM_MSG field returned with the event contains more information. |
| OAMEVN_BOARD_DEAD | 0x40002008 | 1073750024 | An object representing a board is dead. |
| OAMEVN_CREATED | 0x40002003 | 1073750019 | An object was successfully created. |
| OAMEVN_DELETED | 0x40002004 | 1073750020 | An object was successfully deleted. |

| Event name | Hex | Decimal | Description |
|---|---|---|---|
| OAMEVN_FAN_RPM_WARNING | 0x4000200B | 1073750027 | For CG 6565 and CG 6565e boards only.<br>Fan tachometer reaches the warning threshold set in the board configuration file. The event value field contains the fan tachometer, in RPM. |
| OAMEVN_MODIFIED | 0x40002007 | 1073750023 | An object was modified (closed after write access). |
| OAMEVN_RENAMED | 0x40002005 | 1073750021 | An object was successfully renamed. (OAM_MSG contains the old name.) |
| OAMEVN_REPORT | 0x40002002 | 1073750018 | Special internal code used to log report information. |
| OAMEVN_STARTBOARD_DONE | 0x40002101 | 1073750273 | Invocation of **oamStartBoard** successful. The event value field contains a reason code indicating the actual results of the board starting attempt. |
| OAMEVN_STOPBOARD_DONE | 0x40002102 | 1073750274 | Invocation of **oamStopBoard** successful. The event value field contains a reason code indicating the actual results of the board stopping attempt. |
| OAMEVN_STOPBOARD_REQ | 0x40002201 | 1073750529 | A request was made to stop a board with **oamStopBoard**. The board will be stopped in 150 ms. |
| OAMEVN_TEMPERATURE_ALERT | 0x4000200A | 1073750026 | For CG 6565, CG 6565C, and CG 6565e boards only.<br>Board temperature reaches the critical threshold set in the board configuration file. The event value field contains the actual temperature, in tenth of a degree Celsius. |
| OAMEVN_TEMPERATURE_WARNING | 0x40002009 | 1073750025 | For CG 6565, CG 6565C, and CG 6565e boards only.<br>Board temperature reaches the warning threshold set in the board configuration file. The event value field contains the actual temperature, in tenth of a degree Celsius. |
| OAMEVN_TESTBOARD_DONE | 0x40002103 | 1073750275 | Invocation of **oamTestBoard** successful. The event value field contains a reason code indicating the actual results of the board testing attempt. |
| OAMEVN_TRACE | 0x40002006 | 1073750022 | Indicates trace information (potentially high-speed). |

For more information about NMS OAM service event handling, refer to *NMS OAM event structure* on page 27.

## NMS OAM reason codes

Several NMS OAM service functions are asynchronous functions that return a code indicating whether the invocation was successful or not. Subsequent events indicate whether the function completed or not.

The following table alphabetically lists the NMS OAM service reason codes:

| Reason code | Hex | Decimal | Description |
|---|---|---|---|
| OAM_REASON_FAILED | 0x40001001 | 1073745921 | Indicates that a board start, stop, or test attempt failed. |
| OAM_REASON_FINISHED | 0x40001000 | 1073745920 | Indicates that a board was started or stopped successfully, or a board test completed successfully. |

For more information about NMS OAM service event handling, refer to *NMS OAM event structure* on page 27.

# Index