DSM−CC User−User Client Reference Manual

PRELIMINARY DRAFT 2002.06.10

Bionic Buffalo Corporation

Copyright and License

Author and Publisher

Bionic Buffalo Corporation
2533 North Carson Street, Suite 1884
Carson City, Nevada 89706−0147
United States of America

telephone +1 775 882 1842
fax +1 775 882 6047
e−mail *query@tatanka.com*
web *http://www.tatanka.com*

Related Documents and Products

This document pertains to products licensed by Bionic Buffalo Corporation. For other documents and for additional information, see *http://www.tatanka.com/prod/info/dsmcc.html*, or send e−mail to *query@tatanka.com*.

Document Revision History

Original release, 7 August 1997.
Various updates, 1997 to 2002.
Update, 2002.06.10, for A−series software.

Printed 16:40:37, Friday, 21 June, 2002.

Updates and revisions of this document are announced on Bionic Buffalo's DSM−CC e−mail announcement list. See *http://www.tatanka.com/bbc/lists.html* for information.

Additional Document Information

Project name: *spain*.
Document file name: *spain_ref_20020610*.

Written and illustrated using StarOffice and OpenOffice.

Contents

COPYRIGHT AND LICENSE....2
AUTHOR AND PUBLISHER....2
RELATED DOCUMENTS AND PRODUCTS....2
DOCUMENT REVISION HISTORY....2
ADDITIONAL DOCUMENT INFORMATION....3

CONTENTS....4

OVERVIEW....7

LOCAL AND REMOTE OBJECTS....8

TRANSPARENT SERVER INTERFACES....9
THE DOWNLOAD INTERFACES....10
THE SESSION INTERFACES....11
THE SERVICE GATEWAY INTERFACES....11
ABSTRACT (INHERITED) INTERFACES....12
THE EVENT INTERFACE....13
THE KIND INTERFACE....13
THE SECURITY INTERFACE....13
CLIENT IMPLEMENTATIONS OF SERVICE OBJECTS....14
INTERFACES DEFINED BY CORBA....14

COMPATIBILITY, PORTABILITY, AND EXTENSION....15

C LANGUAGE MAPPING....15
MEMORY MANAGEMENT....16
EXTENSIBLE, DISTRIBUTED IMPLEMENTATIONS....17

APPLICATION PROGRAMMER INTERFACE (API)....19

COSNAMING − DATA STRUCTURES FOR THE NAMING SERVICE....20
COSNAMING::BINDINGITERATOR − CAPTURES THE BINDINGS OF A NAMING CONTEXT....23
COSNAMING::BINDINGITERATOR::DESTROY − DISCARD A BINDINGITERATOR....24
COSNAMING::BINDINGITERATOR::NEXT_N − RETURN THE NEXT N BINDINGS OF AN ITERATOR....25
COSNAMING::BINDINGITERATOR::NEXT_ONE − RETURN THE NEXT BINDING OF AN ITERATOR....27
COSNAMING::NAMINGCONTEXT − CONTEXT IN WHICH NAMES ARE BOUND....28
COSNAMING::NAMINGCONTEXT::BIND − CREATE AN NOBJECT BINDING....31
COSNAMING::NAMINGCONTEXT::BIND_CONTEXT − CREATE AN NCONTEXT BINDING....33
COSNAMING::NAMINGCONTEXT::BIND_NEW_CONTEXT − CREATE A NEW CONTEXT AND BIND IT TO A NAME....35
COSNAMING::NAMINGCONTEXT::DESTROY − DELETE A NAMING CONTEXT....37
COSNAMING::NAMINGCONTEXT::LIST − LIST THE BINDINGS OF A CONTEXT....38
COSNAMING::NAMINGCONTEXT::NEW_CONTEXT − CREATE A NEW CONTEXT....40
COSNAMING::NAMINGCONTEXT::REBIND − CREATE AN NOBJECT BINDING....41
COSNAMING::NAMINGCONTEXT::REBIND_CONTEXT − CREATE AN NCONTEXT BINDING....43
COSNAMING::NAMINGCONTEXT::RESOLVE − RETRIEVE AN OBJECT BOUND TO A NAME....45
COSNAMING::NAMINGCONTEXT::UNBIND − REMOVE A BINDING FROM A CONTEXT....46
DSM::ACCESS − COMMON DESCRIPTION AND ACCESS CONTROL ATTRIBUTES....47
DSM::ACCESS::HIST − VERSION AND TIME OF A PERSISTENT OBJECT....48
DSM::ACCESS::LOCK − STATUS OF READ AND WRITE LOCKS....51
DSM::ACCESS::PERMS − ACCESS PERMISSION INFORMATION FOR AN OBJECT....53
DSM::ACCESS::SIZE − SIZE OF AN OBJECT'S ATTRIBUTES....56
DSM::BASE − COMMON OPERATIONS FOR DELETION OF OBJECTS AND TERMINATION OF ACCESS TO OBJECTS....57
DSM::BASE::CLOSE − INDICATE OBJECT ACCESS NO LONGER REQUIRED....58

DSM::BASE::DESTROY − DESTROY AN OBJECT INSTANCE....60
DSM::COMPATIBILITYDESCRIPTOR − INVENTORY OF HARDWARE AND SOFTWARE....61
DSM::COMPOSITE − RELATE CHILD OBJECTS TO A COMMON PARENT....64
DSM::CONFIG − CONTROL INVOCATION BEHAVIOUR....65
DSM::CONFIG::INQUIRE − CHECK STATUS OF DEFERRED OPERATION....67
DSM::CONFIG::WAIT − WAIT UNTIL A DEFERRED OPERATION HAS COMPLETED....68
DSM::DIRECTORY − BINDING NAMES TO OBJECTS OR DATA....69
DSM::DIRECTORY::BIND − BIND OBJECT REFERENCE TO NAME....71
DSM::DIRECTORY::BIND_CONTEXT − BIND DIRECTORY TO NAME....72
DSM::DIRECTORY::BIND_NEW_CONTEXT − CREATE A NEW DIRECTORY AND BIND IT TO A NAME....73
DSM::DIRECTORY::CLOSE − INDICATE DIRECTORY ACCESS NO LONGER REQUIRED....74
DSM::DIRECTORY::DESTROY − DESTROY A DIRECTORY....76
DSM::DIRECTORY::GET − RETURN ATTRIBUTE VALUES BOUND TO A PATH SPECIFICATION....77
DSM::DIRECTORY::HIST − VERSION AND TIME OF A DIRECTORY....79
DSM::DIRECTORY::LIST − RETURN LIST OF BINDINGS....80
DSM::DIRECTORY::LOCK − STATUS OF DIRECTORY READ AND WRITE LOCKS....81
DSM::DIRECTORY::NEW_CONTEXT − CREATE A NEW DIRECTORY....82
DSM::DIRECTORY::OPEN − RESOLVE THE OBJECTS OF A PATH....83
DSM::DIRECTORY::PERMS − ACCESS PERMISSION INFORMATION FOR A DIRECTORY....85
DSM::DIRECTORY::PUT − BIND ATTRIBUTE VALUES TO A PATH SPECIFICATION....86
DSM::DIRECTORY::REBIND − RENAME AN OBJECT....88
DSM::DIRECTORY::REBIND_CONTEXT − RENAME A DIRECTORY....89
DSM::DIRECTORY::RESOLVE − RETURN OBJECT REFERENCE FOR GIVEN NAME....90
DSM::DIRECTORY::SIZE − SIZE OF A DIRECTORY....91
DSM::DIRECTORY::UNBIND − REMOVE NAME FROM AN OBJECT....92
DSM::DOWNLOAD − MANAGE DOWNLOAD CLIENT....93
DSM::DOWNLOAD::ALLOC − ALLOCATE MEMORY BUFFERS FOR A DOWNLOAD....95
DSM::DOWNLOAD::CANCEL − CANCEL A DOWNLOAD IN PROGRESS....97
DSM::DOWNLOAD::INFO − OBTAIN INFORMATION ABOUT PREREQUISITE DOWNLOAD MODULES....98
DSM::DOWNLOAD::START − START DOWNLOAD AND TRANSFER MODULES TO CLIENT....100
DSM::DOWNLOADSI − SERVER INTERFACE FOR DSM::DOWNLOAD OBJECTS....101
DSM::DOWNLOADSI::CANCEL − CANCEL DOWNLOAD IN PROGRESS....102
DSM::DOWNLOADSI::DEINSTALL − UNBIND DOWNLOAD CONFIGURATION FROM SERVER....105
DSM::DOWNLOADSI::INFO − OBTAIN DOWNLOAD INFORMATION AND NEGOTIATE PARAMETERS....106
DSM::DOWNLOADSI::INSTALL − BIND DOWNLOAD CONFIGURATION TO SERVER....108
DSM::DOWNLOADSI::PROCEED − TRANSFER DOWNLOAD DATA BLOCKS....110
DSM::EVENT − RECEIVE EVENTS IN STREAM DATA....112
DSM::EVENT::EVENTLIST − LIST OF AVAILABLE EVENTS....113
DSM::EVENT::NOTIFY − OBTAIN EVENT DATA FROM STREAM EVENT DESCRIPTOR....115
DSM::EVENT::SUBSCRIBE − SUBSCRIBE TO RECEIVE AN MPEG STREAM EVENT....116
DSM::EVENT::UNSUBSCRIBE − END SUBSCRIPTION TO AN MPEG STREAM EVENT....117
DSM::FILE − FILE SYSTEM INTERFACE....118
DSM::FILE::CLOSE − CLOSE A REFERENCE TO A FILE....119
DSM::FILE::CONTENT − CONTENTS OF A FILE....120
DSM::FILE::CONTENTSIZE − SIZE OF FILE CONTENTS....121
DSM::FILE::DESTROY − DESTROY A FILE....122
DSM::FILE::HIST − VERSION AND TIME OF A FILE....123
DSM::FILE::LOCK − STATUS OF FILE READ AND WRITE LOCKS....124
DSM::FILE::PERMS − ACCESS PERMISSION INFORMATION FOR A FILE....125
DSM::FILE::READ − RANDOM ACCESS READ FROM A FILE....126
DSM::FILE::SIZE − SIZE OF A FILE....128
DSM::FILE::WRITE − RANDOM ACCESS WRITE TO A FILE....129
DSM::FIRST − OBTAIN FIRST OBJECTS....131
DSM::INTERFACES − INTERFACE REPOSITORY OPERATIONS....132
DSM::KIND − DETERMINE SUPPORTED INTERFACES....133
DSM::KIND::HAS_A − DETERMINE WHETHER OBJECT SUPPORTS AN INTERFACE....135

DSM::KIND::IS_A − SHOW INTERFACES SUPPORTED BY AN OBJECT....136
DSM::LIFECYCLE − CREATE INTEROPERABLE OBJECT REFERENCE....137
DSM::SECURITY − PROVIDE CREDENTIALS....138
DSM::SECURITY::AUTHENTICATE − REQUEST AUTHENTICATION....139
DSM::SERVICEGATEWAY − ACCESS TO A SERVICE DOMAIN....140
DSM::SERVICEGATEWAYSI − SERVER INTERFACE FOR SERVICE GATEWAY....142
DSM::SERVICEGATEWAYUU − SERVER INTERFACE FOR SERVICE GATEWAY....144
DSM::SESSION − ATTACH TO OR DETACH FROM A SERVICE GATEWAY....146
DSM::SESSION::ATTACH − ATTACH TO A SERVICE GATEWAY DOMAIN....147
DSM::SESSION::DETACH − DETACH FROM A SERVICE GATEWAY DOMAIN....149
DSM::SESSIONSI − SERVER INTERFACE FOR SESSION....150
DSM::SESSIONSI::ATTACH − ATTACH TO A SERVICE GATEWAY DOMAIN....151
DSM::SESSIONSI::DETACH − DETACH FROM A SERVICE GATEWAY DOMAIN....153
DSM::SESSIONUU − INTERFACE TO U−N SESSION PROTOCOL....154
DSM::SESSIONUU::ATTACH − DEFINE UUDATA FOR SESSION ESTABLISHMENT....155
DSM::SESSIONUU::DETACH − DEFINE UUDATA FOR SESSION TEARDOWN....157
DSM::STATE − SUSPEND OR RESUME APPLICATION STATE....159
DSM::STATE::RESUME − RESUME A SERVICE FROM A PREVIOUS APPLICATION STATE....160
DSM::STATE::SUSPEND − SUSPEND APPLICATION STATE FOR A SERVICE....161
DSM::STREAM − THE STREAM INTERFACE....162
THE STREAM STATE MACHINE: SIMPLIFIED VERSION....162
Paused States....*162*
Transporting States....*163*
Searching States....*164*
THE STREAM STATE MACHINE: DETAILED VERSION....164
NORMAL PLAY TIME AND RATE OF PLAY....166
DSM::STREAM::CLOSE − CLOSE A REFERENCE TO A STREAM....167
DSM::STREAM::DESTROY − DESTROY A STREAM....168
DSM::STREAM::HIST − VERSION AND TIME OF A STREAM....169
DSM::STREAM::INFO − STREAM IDENTIFICATION AND CHARACTERISTICS....170
DSM::STREAM::JUMP − WHEN STREAM REACHES STOP NPT, RESUME AT START NPT....172
DSM::STREAM::LOCK − STATUS OF STREAM READ AND WRITE LOCKS....173
DSM::STREAM::PAUSE − STOP SENDING STREAM WHEN NPT IS REACHED....174
DSM::STREAM::PERMS − ACCESS PERMISSION INFORMATION FOR A STREAM....175
DSM::STREAM::PLAY − PLAY STREAM FROM START NPT UNTIL STOP NPT....176
DSM::STREAM::RESET − RESET A STREAM STATE MACHINE....177
DSM::STREAM::RESUME − START SENDING STREAM AT NPT....178
DSM::STREAM::SIZE − SIZE OF A STREAM....179
DSM::STREAM::STATUS − OBTAIN STATUS OF A STREAM....180
DSM::VIEW − RELATIONAL VIEW OF DATA....182

INDEX....183

Overview

This document provides information for use of the DSM−CC User−User client. It includes a description of the API available to applications.

Bionic Buffalo has created various versions of DSM−CC software since 1996. This document is the first released preliminary draft relating to the A−series software versions. The User−User software is now being released in two forms: a commercial version (project "spain") and a free version (project "samoa"). The free version implements a subset of the specification. Each of the API pages has a heading, "Availability". This version of the document does not yet indicate which functions are available in samoa; that will be added in a future release.

This is a **preliminary** release of this document. There is still a list of corrections and additions, so please use with caution.

Enjoy.

Local and Remote Objects

Although DSM−CC is designed for client−server communication, not all of the objects it defines are to be found in the server. Some objects are local objects.

The User−User (U−U) portion of the DSM−CC specification classifies interfaces in several ways. One classification is based on the "origin" of the interface, and results in six categories:

- the U−U RPC library, which defines server interfaces available to client applications by means of remote procedure calls (RPCs)
- session objects, which are essentially data structures carried within the uuData field of U−N session messages
- download objects, which are data structures carried by on−demand U−N download messages
- carousel objects, which are data structures transmitted by cyclic data carousels (using a form of the U−N download protocol)
- local objects, which are defined within the client and not accessible remotely
- stream descriptors, which are data structures included within MPEG or other transport streams

This classification is useful from an architectural standpoint, but does not always help the programmer. For example, a `DSM::File` object is accessed the same way by an application, whether it is transmitted using carousels, on−demand download, or RPC. In IDL terms, the definition of a `DSM::File` object is constant across all these protocols.

Another classification, based on the IDL and consequent C−language mapping seen by applications, distinguishes among three kinds of interfaces:

- the API, or Application Portability Interface, which is that seen by applications (another interpretation of "API" is "application *programming* interface", but DSM−CC uses "portability" as the middle word)
- the SII, or Service Interoperability Interface, which is the interface presented by the server over the wire
- two UU interfaces (origin of name uncertain): `DSM::SessionUU` (implemented by the User−Network client), and `DSM::ServiceGatewayUU` (implemented by the server)

Normally, an application should be written only to the API, and should ignore the SII and UU interfaces. In a well−designed system, the API is the only interface required by almost any application.

Sometimes, this classification results in two or three distinct interfaces having almost the same names. For example, there are `DSM::Session` (API), `DSM::SessionSI` (SII), and `DSM::SessionUU` (UU) interfaces. Although these three session interfaces are related, they are certainly distinct, and an application should not confuse them with one another.

On the other hand, not every interface has three forms. For instance, there is a `DSM::View` interface, but no `DSM::ViewSI` or `DSM::ViewUU` interface defined. Although no separate `DSM::ViewSI` interface is defined, `DSM::ViewSI` is taken to be logically equivalent to `DSM::View`, since the local interface is the same as the over−the−wire interface in this case.

Transparent Server Interfaces

We refer to the definitions of server objects with APIs equivalent to SIIs as *transparent server interfaces*. With such objects, the over−the−wire protocol is the same as the API. There are basically no local semantics. The target object is on the server, and the local API is only a convenient way to communicate with the (remote) server object.

Most DSM−CC User−User interfaces are transparent in this sense. These include the following:

Interface	***Description***
`DSM::Access`	(abstract) version, time, permissions, and size attributes
`DSM::Base`	(abstract) `close` and `destroy` operations
`DSM::BindingIterator`	capture bindings of a context
`DSM::Composite`	(abstract) composite objects
`DSM::Directory`	naming context for attributes and objects
`DSM::File`	traditional data files
`DSM::Interfaces`	access to interface repository
`DSM::State`	(abstract) `suspend` or `resume` application state
`DSM::Stream`	audio and video content streams
`DSM::View`	data seen in relational form

Although these may logically be seen as remote, server objects, there are two possible deviations from this model.

The first deviation allows the client to access the remote server objects by way of a local, proxy object. This is transparent to the application. For example, the application might access a local `DSM::Stream` object, which in turn accesses the remote `DSM::Stream` object. This might be done for implementation convenience, portability, compatibility, or other reasons, but in any case the application sees the same API and semantics.

The second deviation is a modification of the API to permit asynchronous operation in non−threaded environments. The API, as defined by the standard CORBA mapping, blocks until an operation is complete and status has been returned from the server. For instance, a call to the `read` operation of a remote `DSM::File` object will not return until the data have been returned to the caller. For operations which normally would return `void`, the DSM−CC specification allows them instead to return `DSM::RequestHandle`. In such a situation, a local request object is created, and the operation returns immediately to the caller before completion. The request is sent to the server at the time of the call, and the server can process the request while the client application continues. Eventually, the client can interrogate the request object or wait for the server to send its response. In the API, the operation returns `DSM::RequestHandle`, but in the SII, the operation returns `void`. (Refer to the section below on `DSM::Config` for more information.)

The Download Interfaces

The download protocol must be distinguised from `DSM::Download` and `DSM::DownloadSI` objects.

The download protocol, also called User−Network Download, is a distinct DSM−CC protocol with a variety of configuration options. It does not have any awareness of objects, only of data. It can exist in flow−controlled, non−flow−controlled, and carousel implementations. It defines four separate message flows (control up, control down, data up, data down), not all of which are used in every implementation.

Download data, or potential download data which may be carried by one of the forms of the download protocol, may be mapped to objects in several ways, or not mapped at all. When download data is mapped to an object, it may be transparent to the user (so the user is unaware of the data's accessibility using the download protocol), or it may be explicit.

When an object is explicitly associated with the download protocol, its kind is `"DSM::Download"`, or `"dnl"`. The kind can be determined from the `Directory`, or by using the `DSM::Kind` interface. (A given object may inherit the `DSM::Download` interface, while also having some other interface, giving the application a choice of mechanisms to access the object.)

The `DSM::Download` API is used by applications to cause a `DSM::Download` object to be transferred manually to the client. Although the download object is on the server, the API is implemented locally. The server presents a `DSM::DownloadSI` interface, which provides some of the functions needed by the local `DSM::Download` object. The other needed functions are provided by the User−Network Download protocol software.

The Session Interfaces

Interaction between a client and a server begins with establishment of a session. The `DSM::Session` interface defines a local object with two operations, `attach` and `detach`, used to establish and tear down sessions. Establishing a session involves (among other things) finding a server, and obtaining some initial object references. One of the object references returned by `attach` is that of the service gateway object for the session.

Two different protocols may be used between the client and the server for session establishment and teardown. One is the User−Network protocol, and the other is the User−User protocol.

When sessions are managed using the User−Network protocol, the software implementing the User−Network protocol provides a local interface `DSM::SessionUU` to the local `DSM::Session` object. Both objects are local.

When sessions are managed using the User−User protocol, the local `DSM::Session` object invokes a remote server object defined by the `DSM::SessionSI` interface.

In summary, `DSM::Session` and `DSM::SessionUU` are implemented locally, while `DSM::SessionSI` is implemented in the server.

The Service Gateway Interfaces

The `DSM::Session::attach` operation returns (among other things) an object reference for the session's *service gateway*. A service gateway is a local object which inherits the `DSM::Session` and `DSM::Directory` interfaces. It may be viewed as the top−level directory of a server, with the added ability to attach to new servers. (A physical server may implement more than one service gateway, and the implementation of a single service gateway may be the cooperative effort of more than one physical server.)

If the User−Network protocol is used, then the `Session` operations of the local `DSM::ServiceGateway` object rely on the local `DSM::SessionUU` interface provided by the User−Network client code, and the `Directory` operation requests are sent to a remote server `DSM::ServiceGatewayUU` object. `DSM::ServiceGatewayUU` is almost identical to `DSM::Directory` (access control is different).

If the User−Network protocol is not used, then both the `Directory` operations and the `Session` operations of the local `DSM::ServiceGateway` rely on the server's `DSM::ServiceGatewaySI` operations. The local object simply passes all of these calls to the remote server object. `DSM::ServiceGatewaySI` inherits `DSM::Directory` (same on the client as the server) and `DSM::SessionSI` (the server form of the client's `DSM::Session` interface).

In summary, `DSM::ServiceGateway` is implemented on the client, while `DSM::ServiceGatewayUU` and `DSM::ServiceGatewaySI` are implemented on the server.

Abstract (Inherited) Interfaces

Certain interfaces are never instantiated as objects themselves. Instead, they are inherited by other interfaces. For instance, although the `DSM::Base` interface is defined, there is never a `DSM::Base` object *per se*. Other objects (such as `DSM::File` or `DSM::Stream`) inherit the `DSM::Base` interface, and `DSM::Base` operations can be invoked on `DSM::File` or `DSM::Stream` objects.

If an object can be created from an interface, then the interface is said to be *instantiable*. If an object cannot be created from an interface, and the interface can only be inherited, then the interface is said to be *abstract*. An instantiable interface can be inherited, and an abstract interface can inherit from other abstract interfaces.

Inheritance can be part of an object's definition (`DSM::File` *always* inherits `DSM::Base`), or it can be optional (a `DSM::Stream` object may or may not inherit `DSM::Event`).

Usually, if an object inherits an interface, the implementation of the methods is done along with the implementation of the non−inherited methods. However, there are some exceptions, which will be explained below. For instance, consider the `DSM::Base` interface, which defines the `close` and `destroy` operations. If a client object inherits `DSM::Base`, then `close` and `destroy` will be implemented on the client. If a server object inherits `DSM::Base`, then `close` and `destroy` will be implemented on the server.

The following table summarizes the abstract interfaces defined by DSM−CC:

Interface	***Inherited by***	***Description***
`DSM::Access`	any object	version, time, permissions, and size attributes
`DSM::Base`	any object	`close` and `destroy` operations
`DSM::Composite`	any object	composite objects
`DSM::Config`	any object	control invocation behaviour
`DSM::Event`	`DSM::Stream`	events within a stream
`DSM::First`	`DSM::Session`	find first objects of a session
`DSM::Kind`	any object	determine supported interfaces
`DSM::LifeCycle`		
`DSM::Security`	any object	authenticate user
`DSM::State`	any object	`suspend` or `resume` application state

The Event Interface

The `DSM::Event` interface is implemented partly on the server, and partly on the client. There are three operations defined on the `DSM::Event` interface: `subscribe`, `unsubscribe`, and `notify`. The application sees a single API, but each `Event` object's `notify` operation is implemented in the client, while the `subscribe` and `unsubscribe` operations are implemented in the server.

The SII for `DSM::Event` is a subset of the API. The SII omits the `notify` operation.

The Kind Interface

The DSM−CC specification says that `DSM::Kind` provides "local" operations, but describes an interface inherited by all objects. `DSM::Kind` operations `has_a` and `is_a` can be invoked on any object. In practice, these operations are implemented locally (on the client), though conceptually the objects may be local or remote.

The Security Interface

Although the specification treats `DSM::Security` as a transparent interface (with SII the same as API), the Bionic Buffalo implementations allow a separate client `DSM::Security` interfaces.

When keys are kept in a tamper−resistant module such as a smart card, i−button, or other such token, the keys should not be released to the outside. In such cases, there are several components to consider:

1. the application itself, which may or may not provide passwords, PINs, or other authentication data

2. a security object sitting between the application and the token, hiding the token communications fromt he application

3. the security token, which may encrypt or authenticate application data, and which may provide and encrypt or authenticate additional security data kept within the token itself

4. a security object or library routine pseudo−object sitting between the token and the SII, accessible using the IOR of the server object, and merging additional protocol data

5. the SII on the server

Although not every object in this "chain" of objects requires the same `DSM::Security` interface, providing a consistent interface is convenient from an implementation point of view. Therefore, the client implementation may be more complex internally than surface appearances would indicate.

Client Implementations of Service Objects

Although `DSM::File` and `DSM::Directory` interfaces are intended for use on server−based objects, Bionic Buffalo client implementations allow them to be hosted on servers. This makes the client directory and file system available to client applications, and also (if properly configured) to servers or to other clients.

With the same API used for local and remote file systems, applications can be written so that changing from one object to another is transparent. For example, a diskless system might save files on a server, until a client upgrade makes local storage possible. In that case, the application would not need to change: the same file API would be used in both cases. In another case, if the client were to export the `DSM::File` interface over the network, then an e−mail application running on a server could deliver mail to a client disk file or to a server disk file using the same interface in both instances.

It is also possible to install other server software on clients, for peer−to−peer distribution of content, distributed databases, and other applications.

Interfaces Defined by CORBA

Although they are not mentioned by the DSM−CC specification, some operations are defined by the `CORBA::Object` interface, and are (or should be) inherited by every object. These are described in the API section, below. In most cases, the inherited operations are implemented in the same location as the explicitly−defined operations. There are some exceptions (such as `CORBA::Object::duplicate`) which are implemented locally regardless of the object's primary location.

Compatibility, Portability, and Extension

Most DSM−CC systems are not CORBA−compliant. The DSM−CC specification seems to anticipate this, since it defines interfaces (such as `DSM::Kind`) that are redundant in the context of interface repositories and the other accoutrements of a complete CORBA implementation.

On the other hand, the DSM−CC specification isn't complete enough to describe fully−portable applications. For instance, there is no discussion of how to initialize the ORB, or how to obtain the first objects (such as `DSM::Session`) that are needed to write a DSM−CC application. The discussion about `close` operations, which purportedly should be used to destroy object references, is ambiguous and seems to conflict with CORBA anyway.

Bionic Buffalo has used three simple rules in designing its DSM−CC implementations:

1. Where the CORBA specifications conflict with the DSM−CC specification, CORBA takes precedence.

2. Resolve ambiguities and implement missing or incomplete definitions using CORBA tools.

3. If the "real world" doesn't interoperate with the result of the above rules, provide an altternative workaround, but make the deviation from the standards clear and explicit.

The rest of this section describes some of the main results of applying these rules.

C Language Mapping

The DSM−CC specification was written with reference to early versions of the CORBA IDL−to−C mapping. The most striking change in subsequent versions of the mapping was in the placement of the CORBA Environment in a function's calling sequence.

In CORBA 1.x, an IDL operation

```
interface iface { int op1 ( in int a1, out int a2 ); } ;
```

was mapped to a C function as

```
typedef CORBA_Object iface ;
int iface_op1 ( iface o, CORBA_Environment * ev, int a1, int * a2 ) ;
```

In CORBA 2.x, the same operation is mapped to

```
typedef CORBA_Object iface ;
int iface_op1 ( iface o, int a1, int * a2, CORBA_Environment * ev ) ;
```

The difference is in the position of the CORBA_Environment parameter: it used to be second, now it is always last. The DSM−CC specification follows the older mapping, while Bionic Buffalo implementations use the newer mapping.

This change does not affect the protocol with the server, only the API.

The reasoning behind this choice was mostly the same as for similar decisions involving other conflicts between the CORBA and DSM−CC specifications: to decide otherwise would make interoperability with other CORBA tools and applications more difficult, there was very little legacy C code to convert, and at any rate the incompleteness of the DSM−CC specification requires reliance on the CORBA specifications to fill in the gaps.

Memory Management

The C mapping carefully distinguishes CORBA memory from "ordinary" memory. This is done for two reasons: an ORB may want to allocate memory from a special region (so it is accessible to network drivers, for instance), and CORBA data structures sometimes have hidden fields (such as release flags).

A sequence defined in IDL is mapped to a C data structure, but it is improper to allocate the sequence using `malloc()`. Instead, each sequence `ABC` has associated allocation routines `ABC__alloc()` and `ABC_allocbuf()`. A CORBA implementation may add extra fields or allocate all CORBA structures from some special pool, so `malloc()` isn't adequate. Likewise, any CORBA memory which is no longer needed, must be released using `CORBA_free()`, not `free()`.

In the C mapping, an object reference is mapped to a `void *` pointer, with the assumption that this pointer refers to an opaque data item. Duplication of the pointer requires duplicating the opaque data item, and `CORBA_Object_duplicate()` is provided for that purpose. When the object reference is no longer required, `CORBA_Object_release()` may be used to free the opaque data structure. (An alternative is to share a data structure in user address space, maintaining a usage counter for the number of references.)

In a simple client without memory mapping, it is possible to implement CORBA so that each reference to a given object points to the same system data structure, and `CORBA_Object_duplicate()` would be implemented merely by returning its first argument. That seems to be the kind of implementation the writers of DSM−CC envisioned, when they made statements such as "Close means delete the object references and therefore the Client's ability to communicate with the object." Such an approach doesn't always work in a more complex, memory mapped environment where each object reference points to a separate data structure: deleting one structure leaves the others allocated.

The DSM−CC specification makes little mention of these rules for managing memory. It appears to envision a simple client with few resources, a basic RTOS, and no file system: an extension of the low−end set top box in common use for cable and satellite systems. Such a specification is not a bad thing, because it allows the possibility of hosting DSM−CC on low−end systems. However, if applications written for those low−end systems ignore the discipline required for tasks such as memory management, those applications will not be completely portable to more complex environments such as PC−based clients.

Bionic Buffalo takes the position that low−end implementations which adhere to the complete rule set for CORBA are not necessarily inefficient, and do not need excessive resources. In the example above, it is possible to have

```
#define CORBA_Object_duplicate(o,ev) (o)
```

on a low−end implementation, with a library procedure on a high−end implementation. In either case, the programmer can write

```
newref = CORBA_Object_duplicate (oldref, ev) ;
```

In a low−end environment, no inefficiencies are introduced. When an application is moved to a more complex environment, no memory leaks or dangling pointers will occur in this situation.

The result for the programmer is that some operations (such as `DSM::Base::close`) in Bionic Buffalo's DSM−CC may not do the same things that are done by other implementations. (That is, Bionic Buffalo expects a separate subsequent call to `CORBA::Object::release`.) Close reading of the specification doesn't show that either implementation choice violates the rules, only that the DSM−CC document is ambiguous and incomplete. The consequence is reduced portability for applications.

Extensible, Distributed Implementations

The DSM−CC specification does not require that a client or server be constructed using ORBs, object adaptors, interface repositories, and other standard features of a CORBA implementation. Nevertheless, Bionic Buffalo has chosen to base its DSM−CC on CORBA. This allows extensible and highly distributed implementations.

Existing interfaces may be extended, or new ones created, by changing or adding IDL definitions, then writing code for the new methods. No new protocols are needed for this: the interfaces are available automatically by using the CORBA infrastructure.

> *Example:* A new server object might be defined to provide account information to customers. The new object would be accessible to a client application using a local (client) API.

Objects (including client objects) can be accessible over the network, allowing remote management, support, and diagnostic features to be built into a system.

> *Example:* A "self test" object might be installed in the client, accessible to support teams.

> *Example:* In a disk−based client system, software could be updated remotely using the same `DSM::File` interface defined for server access.

Common Java tools and software can relatively easily interoperate (locally or by network) with CORBA−based systems.

> *Example:* A client−based Java application might invoke methods on server and client files, using the `DSM::File` interface.

The components of a server or client can be distributed transparently across multiple processors, even when the CPUs and operating systems are dissimilar.

> *Example:* A `DSM::View` object might be implemented on a database server, a `DSM::Stream` object on a video server, and other objects on a third server.
>
> *Example:* On the client, the `DSM::Security` object might be implemented within a separate, tamper−resistant module.

Application Programmer Interface (API)

This chapter describes the object interfaces defined for DSM−CC User−User clients. It includes descriptions of SII and UU interfaces, as well as interfaces to client library objects.

CosNaming − data structures for the naming service

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    CosNaming_BindingIterator ;
typedef CORBA_Object                    CosNaming_NamingContext ;
typedef CORBA_Object                    CosNaming_NamingContextExt ;

typedef CORBA_string                    CosNaming_Istring ;

typedef struct CosNaming_NameComponent
        {
        CosNaming_Istring               id ;
        CosNaming_Istring               kind ;
        }
        CosNaming_NameComponent ;

typedef struct CosNaming_Name
        {
        CORBA_unsigned_long             _maximum ;
        CORBA_unsigned_long             _length ;
        CosNaming_NameComponent       * _buffer ;
        }
        CosNaming_Name ;

CosNaming_Name                * CosNaming_Name__alloc ( void ) ;
CosNaming_NameComponent       * CosNaming_Name_allocbuf
                                        ( CORBA_unsigned_long len ) ;

typedef CORBA_enum                      CosNaming_BindingType ;

#define CosNaming_nobject               0UL
#define CosNaming_ncontext              1UL

typedef struct CosNaming_Binding
        {
        /*
         *      Note: In struct Binding, binding_name is incorrectly
         *      defined as a Name instead of a NameComponent. This
         *      definition is unchanged for compatibility reasons.
         */
        CosNaming_Name                  binding_name ;
        CosNaming_BindingType           binding_type ;
        }
        CosNaming_Binding ;

typedef struct CosNaming_BindingList
        {
        CORBA_unsigned_long             _maximum ;
        CORBA_unsigned_long             _length ;
```

```
        CosNaming_Binding             * _buffer ;
        }
        CosNaming_BindingList ;


CosNaming_BindingList
              * CosNaming_BindingList__alloc ( void ) ;
CosNaming_Binding
              * CosNaming_BindingList_allocbuf
                                      ( CORBA_unsigned_long   len ) ;
```

Description

Data structures for the naming service.

The naming service associates names with objects. A name is bound to an object relative to a *naming context*, which is itself an object. Since a naming context is an object, it can also be named.

Three classes are defined in the `CosNaming` module: `NamingContext` (analagous to a directory or folder in traditional filesystems), `BindingIterator` (an ephemeral object used to determine the bindings of a `NamingContext`), and `NamingContextExt` (an extended form of `NamingContext` with operations for stringified names and URLs).

Notes

1. A `CosNaming::NameComponent` has two parts, the `id` and the `kind`, which are both Latin−1 (ISO 8859−1) strings. For two `NameComponents` to be equal, both `ids` and `kinds` must be identical; comparisons are case−sensitive. Implementations may place restrictions on name components, such as length limitations or restrictions to certain characters.

2. *(DSM−CC only)* The `kind` is either a string constructed of a module name and an interface (such as `"DSM::Stream"`), or a string consisting of an interface alias (such as `"str"`). See `DSM::Directory` for more information.

3. A `CosNaming::Name` consists of a sequence of components. The most common interpretation is to view a `CosNaming::Name` as a path name, althought there are other possible semantics. In this interpretation, a naming context is roughly equivalent to a directory. When `CosNaming::Names` correspond to path names, the separator characters of the path names are not included in the `CosNaming::Name` structure, and all of the components except the last must be directories. An implementation may limit the number of components in the sequence.

4. The value of `CosNaming::BindingType` is either `CosNaming::nobject` or `CosNaming::ncontext`. In the most common interpretation, each components of a path name is either an object or a directory.

5. *(DSM−CC only)* DSM−CC extends the use of `Names` to assign attributes to objects.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001

CosNaming::BindingIterator − captures the bindings of a naming context

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    CosNaming_BindingIterator ;
```

Description

A BindingIterator is created by a NamingContext::list operation. It is used to retrieve the bindings of the NamingContext, using its next_one and next_n operations. Once the bindings are retrieved, the BindingIterator::destroy operation is used to discard the BindingIterator.

Notes

1. If a NamingContext changes between two BindingIterator operations, then the results of the second and subsequent BindingIterator operations are undefined by the specification, and thus dependent upon the implementation.

2. Implementations may use a garbage collection mechanism to control the use of resources occasioned by undestroyed BindingIterator objects. Without garbage collection, if applications were not to invoke the destroy operation as appropriate, then the number of BindingIterators could grow without limit. If a BindingIterator operation is called after the iterator is destroyed, or after garbage collection has destroyed the iterator, then the CORBA::OBJECT_NOT_EXIST exception will be raised.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.3.

CosNaming::BindingIterator::destroy − discard a BindingIterator

Synopsis − C

```
#include    <dsmcc.h>

void            CosNaming_BindingIterator_destroy
                        ( CosNaming_BindingIterator     o,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the iterator to be discarded
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

(standard)

Description

Discards a BindingIterator.

Once the bindings have been retrieved from the iterator (by use of next_n and next_one), this operation is used to discard the iterator and any associated resources.

Notes

(none)

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.3.1.3.

CosNaming::BindingIterator::next_n − return the next n bindings of an iterator

Synopsis − C

```
#include    <dsmcc.h>

CORBA_boolean   CosNaming_BindingIterator_next_n
                        ( CosNaming_BindingIterator     o,
                        CORBA_unsigned_long             how_many,
                        CosNaming_BindingList        ** bl,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the iterator to be discarded
how_many	*(in)* the maximum number of bindings to return
bl	*(out)* the returned bindings
ev	*(in/out)* CORBA environment

Returns

CORBA_TRUE, if bl is not empty
CORBA_FALSE, if bl is a zero−length sequence

Exceptions

(standard)

Description

Retrieves a specified number of bindings from an iterator.

This operation and next_one are called repeatedly until all required bindings are retrieved, then destroy is invoked.

Notes

1. If there are no more bindings, then a zero−length sequence is returned. Although the sequence has no elements, the sequence structure itself has been allocated and must be released.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.3.1.2.

CosNaming::BindingIterator::next_one − return the next binding of an iterator

Synopsis − C

```
#include    <dsmcc.h>

CORBA_boolean   CosNaming_BindingIterator_next_one
                        ( CosNaming_BindingIterator     o,
                        CosNaming_Binding            ** b,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the iterator to be discarded
b	*(out)* the returned binding
ev	*(in/out)* CORBA environment

Returns

CORBA_TRUE, if b is not empty (*b != NULL)
CORBA_FALSE, if b is a empty (*b == NULL)

Exceptions

(standard)

Description

Retrieves the next binding from an iterator.

This operation and next_n are called repeatedly until all required bindings are retrieved, then destroy is invoked.

Notes

(none)

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.3.1.1.

CosNaming::NamingContext − context in which names are bound

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    CosNaming_NamingContext ;

typedef CORBA_enum      CosNaming_NamingContext_NotFoundReason ;

#define CosNaming_NamingContext_missing_node            0UL
#define CosNaming_NamingContext_not_context             1UL
#define CosNaming_NamingContext_not_object              2UL

/* CosNaming::NamingContext::NotFound exception */
typedef struct CosNaming_NamingContext_NotFound
        {
        CosNaming_NamingContext_NotFoundReason  why ;
        CosNaming_Name                          rest_of_name ;
        }
        CosNaming_NamingContext_NotFound ;
#define ex_CosNaming_NamingContext_NotFound             \
        "omg.org:CosNaming/NamingContext/NotFound:1.0"

/* CosNaming::NamingContext::CannotProceed exception */
typedef struct CosNaming_NamingContext_CannotProceed
        {
        CosNaming_NamingContext                 cxt ;
        CosNaming_Name                          rest_of_name ;
        }
        CosNaming_NamingContext_CannotProceed ;
#define ex_CosNaming_NamingContext_CannotProceed        \
        "omg.org:CosNaming/NamingContext/CannotProceed:1.0"

/* CosNaming::NamingContext::InvalidName exception */
typedef struct CosNaming_NamingContext_InvalidName
        {
        int                                     filler ;
        }
        CosNaming_NamingContext_InvalidName ;
#define ex_CosNaming_NamingContext_InvalidName          \
        "omg.org:CosNaming/NamingContext/InvalidName:1.0"

/* CosNaming::NamingContext::AlreadyBound exception */
typedef struct CosNaming_NamingContext_AlreadyBound
        {
        int                                     filler ;
        }
        CosNaming_NamingContext_AlreadyBound ;
#define ex_CosNaming_NamingContext_AlreadyBound         \
        "omg.org:CosNaming/NamingContext/AlreadyBound:1.0"
```

```
/* CosNaming::NamingContext::NotEmpty exception */
typedef struct CosNaming_NamingContext_NotEmpty
        {
        int                                     filler ;
        }
        CosNaming_NamingContext_NotEmpty ;
#define ex_CosNaming_NamingContext_NotEmpty             \
        "omg.org:CosNaming/NamingContext/NotEmpty:1.0"
```

Description

A NamingContext is a object which contains a set of name−to−object associations.

An object may have no name, one name, or more than one name. However, no two objects may have the same name (identical id and kind) in the same NamingContext. A NamingContext, being itself an object, may also be named.

Ten operations are defined on NamingContexts: five to create and destroy bindings (bind, bind_context, rebind, rebind_context, unbind); two (resolve, list) to retrieve existing bindings; and three (new_context, bind_new_context, destroy) to create and destroy contexts. Five exceptions (listed above and described in the Notes, below) are defined for these operations, although not every operation raises the same exceptions.

Notes

1. A NamingContext may be viewed as an extensions of the traditional concept of a directory or folder, which is a set of name−to−file associations.

2. *(DSM−CC only)* DSM::Directory inherits CosNaming::NamingContext, and extends the class by adding four new operations and additional data structures.

3. The NotFound exception is raised when an operation cannot find a binding for a name component, or when the binding is inappropriate for the operation. The why member explains the reason, and rest_of_name contains the NameComponents (if any) subsequent to the one causing the exception.

   If why == missing_node, then the first NameComponent in rest_of_name was not bound in its parent context.

   If why == not_context, then the first NameComponent in rest_of_name denoted a binding of type nobject, while the type should have been ncontext.

   If why == not_object, then the first NameComponent in rest_of_name denoted a binding of type ncontext, while the type should have been nobject.

4. The `CannotProceed` exception is raised when the implementation is not able to continue past one of the contexts in a `Name`. The `cxt` member gives the failing context, and the `rest_of_name` member gives the subsequent `NameComponents`. A common reason for this exception is a security violation, where a resolve of `rest_of_name` was not authorized in `ctx`.

5. The `InvalidName` exception is raised when a `Name` is invalid. One kind of invalid name is one of zero length (without any `NameComponents`), but an implementation may define additional reasons to render a `Name` unacceptable.

6. The `AlreadyBound` exception is raised when an attempt is made to create a binding using a `Name` which is already bound. No two `Names` in a `NamingContext` may be the same.

7. The `NotEmpty` exception is raised by the `destroy` operation, when the `NamingContext` contains bindings. All of the bindings must be destroyed before the `destroy` operation is invoked.

8. A `CosNaming::NamingContext` object is created by the `new_context` or `bind_new_context` operation on an existing `CosNaming` object. The initial `CosNaming` object (from which all others may be created) is obtained using `CORBA::ORB::resolve_initial_references`, by supplying an `ObjectID` of `"NameService"`.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.

CosNaming::NamingContext::bind − create an nobject binding

Synopsis − C

```
#include    <dsmcc.h>

void            CosNaming_NamingContext_bind
                        ( CosNaming_NamingContext       o,
                        CosNaming_Name                * n,
                        CORBA_Object                  * obj,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the context in which the binding is to take place
n	*(in)* the name to be bound
obj	*(in)* the object to be bound
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

```
CosNaming::NamingContext::NotFound
CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::InvalidName
CosNaming::NamingContext::AlreadyBound
```

Description

Create an nobject binding in a NamingContext. The name must not already be bound in the context.

Notes

1. If the name is already bound in the context, then the AlreadyBound exception will be raised.

2. An implementation may limit the number of bindings in a context. If this operation would exceed that limit, then the IMP_LIMIT exception will be raised.

3. The binding will survive the destruction of obj. If an object is destroyed, any bindings of which it is a part should be destroyed first.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.5.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.3.1.

CosNaming::NamingContext::bind_context − create an ncontext binding

Synopsis − C

```
#include    <dsmcc.h>

void            CosNaming_NamingContext_bind_context
                        ( CosNaming_NamingContext       o,
                        CosNaming_Name                * n,
                        CosNaming_NamingContext         nc,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the context in which the binding is to take place
n	*(in)* the name to be bound
nc	*(in)* the context object to be bound
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

CosNaming::NamingContext::NotFound
CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::InvalidName
CosNaming::NamingContext::AlreadyBound

Description

Create an ncontext binding in a NamingContext. The name must not already be bound in the context.

Notes

1. If the name is already bound in the context, then the AlreadyBound exception will be raised.

2. An implementation may limit the number of bindings in a context. If this operation would exceed that limit, then the IMP_LIMIT exception will be raised.

3. If nc is NIL, then a BAD_PARAM exception is raised.

4. The binding will survive the destruction of nc. If an object is destroyed, any bindings of which it is a part should be destroyed first.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.6.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.3.3.

CosNaming::NamingContext::bind_new_context − create a new context and bind it to a name

Synopsis − C

```
#include    <dsmcc.h>

CosNaming_NamingContext
                CosNaming_NamingContext_bind_new_context
                        ( CosNaming_NamingContext       o,
                        CosNaming_Name                * n,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* any existing context
n	*(in)* the name to which the new context will be bound
ev	*(in/out)* CORBA environment

Returns

the newly−created NamingContext object

Exceptions

```
CosNaming::NamingContext::NotFound
CosNaming::NamingContext::AlreadyBound
CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::InvalidName
```

Description

Create a new NamingContext object, and bind the new context to a name in the creating context.

Notes

1. The original context, from which all others may be created, is obtained by calling CORBA::ORB::resolve_initial_references with an ObjectId of "NameService".

2. If the implementation limits the number of NamingContext objects or the number of bindings in any single context, and if this operation would exceed either of those limits, then the IMP_LIMIT exception will be raised.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.11.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.6.2.

CosNaming::NamingContext::destroy − delete a naming context

Synopsis − C

```
#include    <dsmcc.h>

void            CosNaming_NamingContext_destroy
                        ( CosNaming_NamingContext       o,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the context to be destroyed
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

CosNaming::NamingContext::NotEmpty

Description

Delete a NamingContext object.

Notes

1. If the context contains bindings, then the NotEmpty exception is raised.

2. If the context is bound to names in other contexts, then those bindings will survive the destruction of this context. However, they will refer to a nonexistent object. Such bindings should be destroyed before destroying the context to which they refere.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.12.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.7.

CosNaming::NamingContext::list − list the bindings of a context

Synopsis − C

```
#include    <dsmcc.h>

void            CosNaming_NamingContext_list
                        ( CosNaming_NamingContext       o,
                        CORBA_unsigned_long             how_many,
                        CosNaming_BindingList        ** bl,
                        CosNaming_BindingIterator     * bi,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the context whose bindings are to be listed
how_many	*(in)* the maximum number of bindings to be listed
bl	*(out)* the list of bindings
bi	*(out)* an iterator to list additional bindings
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

(standard exceptions)

Description

List the bindings of a context. A maximum how_many bindings are listed. If the context has more than how_many bindings, then a BindingIterator is created to list the additional bindings.

Notes

1. If how_many = 0, then bl will have no elements, and all bindings will be captured in bi.

2. An implementation may return less than how_many bindings in bl (if how_many > 0). However, an empty list will not be returned unless how_many = 0 or there are no bindings in the context.

3. If bi is NIL, then all bindings (if any) have been returned in bl.

4. If bi is not NIL, then bi may or may not have any additional bindings. In other words, bi might be empty.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.3.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.8.

CosNaming::NamingContext::new_context − create a new context

Synopsis − C

```
#include    <dsmcc.h>

CosNaming_NamingContext
                CosNaming_NamingContext_new_context
                        ( CosNaming_NamingContext       o,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* any existing context
ev	*(in/out)* CORBA environment

Returns

the newly−created NamingContext object

Exceptions

(standard)

Description

Create a new NamingContext object. The new context is not bound to any name.

Notes

3. Although the new context is created using an existing context, it is not bound in the creating context. Any context may be used to create any other context in this way. The original context, from which all others may be created, is obtained by calling CORBA::ORB::resolve_initial_references with an ObjectId of "NameService".

4. If the implementation limits the number of NamingContext objects, and if this operation would exceed that limit, then the IMP_LIMIT exception will be raised.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.10.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.6.1.

CosNaming::NamingContext::rebind − create an nobject binding

Synopsis − C

```
#include    <dsmcc.h>

void            CosNaming_NamingContext_rebind
                        ( CosNaming_NamingContext       o,
                        CosNaming_Name                * n,
                        CORBA_Object                  * obj,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the context in which the binding is to take place
n	*(in)* the name to be bound
obj	*(in)* the object to be bound
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

CosNaming::NamingContext::NotFound
CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::InvalidName

Description

Create an nobject binding in a NamingContext. Either the name is not already bound in the context, or it is already bound in the context with a binding type of nobject.

Notes

1. If the name is already bound in the context with a binding type of ncontext, then a NotFound exception is raised with why = not_object. In such a case, the rest_of_name member will contain a single component.

4. The binding will survive the destruction of obj. If an object is destroyed, any bindings of which it is a part should be destroyed first.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.7.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.3.2.

CosNaming::NamingContext::rebind_context − create an ncontext binding

Synopsis − C

```
#include    <dsmcc.h>

void            CosNaming_NamingContext_rebind_context
                        ( CosNaming_NamingContext       o,
                        CosNaming_Name                * n,
                        CosNaming_NamingContext         nc,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the context in which the binding is to take place
n	*(in)* the name to be bound
nc	*(in)* the context object to be bound
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

CosNaming::NamingContext::NotFound
CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::InvalidName

Description

Create an ncontext binding in a NamingContext. Either the name is not already bound in the context, or it is already bound in the context with a binding type of ncontext.

Notes

1. If the name is already bound in the context with a binding type of nobject, then a NotFound exception is raised with why = not_context. In such a case, the rest_of_name member will contain a single component.

2. The binding will survive the destruction of nc. If an object is destroyed, any bindings of which it is a part should be destroyed first.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.8.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.3.4.

CosNaming::NamingContext::resolve − retrieve an object bound to a name

Synopsis − C

```
#include    <dsmcc.h>

CORBA_Object    CosNaming_NamingContext_resolve
                        ( CosNaming_NamingContext       o,
                        CosNaming_Name                * n,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the context in which the name is bound
n	*(in)* the name bound to the object
ev	*(in/out)* CORBA environment

Returns

the object bound to the name

Exceptions

```
CosNaming::NamingContext::NotFound
CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::InvalidName
```

Description

Retrieve an object bound to a name in a context.

Notes

1. Resolution of a compound name may involve traversal of multiple naming contexts.

2. The application is responsible for narrowing the class of the returned object. The type of the object is not returned by this operation.

3. The name must match the bound name exactly. There are no wild card or approximate matches.

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.4.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.4.

CosNaming::NamingContext::unbind − remove a binding from a context

Synopsis − C

```
#include    <dsmcc.h>

void            CosNaming_NamingContext_unbind
                        ( CosNaming_NamingContext       o,
                        CosNaming_Name                * n,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the context in which the name is bound
n	*(in)* the name bound in the context
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

```
CosNaming::NamingContext::NotFound
CosNaming::NamingContext::CannotProceed
CosNaming::NamingContext::InvalidName
```

Description

Remove a binding from a context.

Notes

(none)

Availability

greece, spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.9.

OMG 01−02−65, *Naming Service Specification*, Revised Edition, February 2001, Section 2.2.5.

DSM::Access − common description and access control attributes

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_Access ;
```

Description

The DSM::Access object defines common description and access control attributes for other DSM−CC objects.

Notes

1. A DSM::Access object is an abstract object, and is not instantiable. It is defined so that its data structures and attributes may be inherited by other objects.

2. DSM−CC object attributes are accessible through the usual operations (as defined by the IDL−to−C mapping). They are also accessible using DSM::Directory::put and DSM::Directory::get. Conceptually, they are stored in the Directory, although the implementation may chose some different mechanism to achieve the same results.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.2.

DSM::Access::Hist − version and time of a persistent object

Synopsis − C

```
#include    <dsmcc.h>

typedef struct DSM_Version
        {
        CORBA_char                      aMajor ;
        CORBA_char                      aMinor ;
        }
        DSM_Version ;

typedef struct DSM_DateTime
        {
        DSM_s_long      tm_sec ;    /* secs after the minute, 0-61 */
        DSM_s_long      tm_min ;    /* minutes after the hour, 0-59 */
        DSM_s_long      tm_hour ;   /* hours since midnight, 0-23 */
        DSM_s_long      tm_mday ;   /* day of the month, 1-31 */
        DSM_s_long      tm_mon ;    /* months since January, 0-11 */
        DSM_s_long      tm_year ;   /* years since 1900 */
        DSM_s_long      tm_wday ;   /* days since Sunday, 0-6 */
        DSM_s_long      tm_yday ;   /* days since Jan 1, 0-365 */
        DSM_s_long      tm_isdst ;  /* daylight savings time flag */
        }
        DSM_DateTime ;

typedef struct DSM_Access_Hist_T
        {
        DSM_Version     aVersion ;      /* object version */
        DSM_DateTime    aDateTime ;     /* time created or last
                                           updated, GMT */
        }
        DSM_Access_Hist_T ;

#define DSM_Access_Hist_get_ACR         DSM_READER
#define DSM_Access_Hist_put_ACR         DSM_BROKER

DSM_Access            * DSM_Access__get_Hist
                                ( DSM_Access            o,
                                CORBA_Environment     * ev ) ;
void                    DSM_Access__set_Hist
                                ( DSM_Access            o,
                                DSM_Access_Hist_T     * Hist,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object whose history is to be returned or modified
Hist	*(in)* the new update time and version of the object
ev	*(in/out)* the CORBA environment

Returns

`DSM_Access__get_Hist` returns the `Hist` attribute of an object.
`DSM_Access__set_Hist` returns `void`.

Exceptions

(standard)

Description

Returns or sets the version and time attribute of an object.

Notes

1. The range of `0..61` for `tm_sec` allows for the occasional leap second or double leap second.

2. The `DSM_DateTime` structure was modified from the ANSI C standard. It is not, in general, identical to the C structure `tm`.

3. The field `tm_isdst` of `DSM_DateTime` indicates the use of Daylight Savings Time (summer time). The value of the field is defined as follows:

`tm_isdst > 0`	daylight savings time in effect
`tm_isdst = 0`	daylight savings time not in effect
`tm_isdst < 0`	information not available

4. The time and date are specified to be always in terms of GMT (Greenwich Mean Time.) GMT is not affected by daylight savings (summer) time. When summer time is effective in the UK, the designation "BST" (British Summer Time) is used, but the beginning and ending dates are not in general the same as for daylight savings or summer time in other time zones. Athough the `tm_isdst` field is defined by this specification, the purpose is not clear except perhaps for dubious compatibility reasons.

5. `DSM_Access__get_Hist` requires `READER` privilege. `DSM_Access__set_Hist` requires `BROKER` privilege.

1. Attributes also may be set using `DSM::Directory::put`, and read using `DSM::Directory::get`.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.2.

DSM::Access::Lock − status of read and write locks

Synopsis − C

```
#include    <dsmcc.h>

typedef struct DSM_Access_Lock_T
        {
        CORBA_boolean                   readLock ;
        CORBA_boolean                   writeLock ;
        }
        DSM_Access_Lock_T ;

#define DSM_Access_Lock_get_ACR         DSM_READER
#define DSM_Access_Lock_put_ACR         DSM_WRITER

DSM_Access_Lock_T     * DSM_Access__get_Lock
                                ( DSM_Access            o,
                                CORBA_Environment     * ev ) ;
void                    DSM_Access__set_Lock
                                ( DSM_Access            o,
                                DSM_Access_Lock_T     * Lock,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object whose locks are to be returned or set
Lock	*(in)* the new value of the locks
ev	*(in/out)* CORBA environment

Returns

DSM_Access__get_Lock returns DSM_Access_Lock_T
DSM_Access__set_Lock returns void

Exceptions

(standard)

Description

Returns or sets the lock attribute of an object.

Notes

1. DSM_Access__get_Lock requires READER privilege. DSM_Access__set_Lock requires WRITER privilege.

2. Attributes also may be set using DSM::Directory::put, and read using DSM::Directory::get.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.2.

DSM::Access::Perms − access permission information for an object

Synopsis − C

```
#include    <dsmcc.h>

typedef struct DSM_Access_Perms_T
        {
        DSM_u_short                     managerPerm ;
        DSM_u_short                     brokerPerm ;
        DSM_u_longlong                  writerPerm ;
        DSM_u_longlong                  readerPerm ;
        DSM_opaque                      owner ;
        CORBA_string                    aPassword ;
        DSM_opaque                      authData ;
        CORBA_boolean                   allSecure ;
        }
        DSM_Access_Perms_T ;

#define DSM_Access_Perms_get_ACR        DSM_OWNER
#define DSM_Access_Perms_put_ACR        DSM_OWNER

DSM_Access_Perms_T    * DSM_Access__get_Perms
                                ( DSM_Access            o,
                                CORBA_Environment     * ev ) ;
void                    DSM_Access__set_Perms
                                ( DSM_Access            o,
                                DSM_Access_Perms_T    * Perms,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object whose permissions are to be set or returned
Perms	*(in)* the new permissions
ev	*(in/out)* CORBA environment

Returns

DSM_Access__get_Perms returns DSM_Access_Perms_T
DSM_Access__set_Perms returns void

Exceptions

(standard)

Description

Returns or sets the permission attribute of an object.

Notes

1. Both of these operations require OWNER privilege.

2. managerPerm, brokerPerm, writerPerm, and readerPerm are bit masks specifying which manager, broker, writer, and reader groups have access to the object. A user may belong to one or more manager groups, broker groups, writer groups, and reader groups; the mapping is made when the user attaches to a ServiceGateway.

3. Each DSM−CC operation requires a certain role: OWNER, MANAGER, BROKER, WRITER, or READER. An operation is allowed only if the user belongs to one of the groups associated with the necessary role. The required roles for an operation X is indicated in this documentation for each operation, or is shown in the header files by a definition such as `const /* IDL*/ AccessRole X_ACR = Y;` or `/* C */ #define X_ACR Y`.

4. The privileges of each authorized group are summarized as follows:

Function	Manager	Owner	Broker	Writer	Reader
read	yes	yes	yes	yes	yes
write or update	yes	yes	yes	yes	no
authorize access by a client	yes	yes	yes	no	no
create or destroy	yes	yes	no	no	no
full system administration authority	yes	no	no	no	no

5. The owner field defines the object's creator. When an object is created, its initial owner is the user which created the object.

6. An implementation may or may not map Principal to user or to owner. A different mechanism may be defined for user identification.

7. If aPassword is non−null, then a password must be provided for access. If authData is non−null, then an implementation−dependent encryption challenge must be met before access is permitted. Refer to the discussion of security and authentication in the notes on implementation and use for more information.

8. If allSecure is true, then all lower communication layers must use encryption when transferring any method parameters for this object.

9. Attributes also may be set using DSM::Directory::put, and read using DSM::Directory::get.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.2.

DSM::Access::Size − size of an object's attributes

Synopsis − C

```
#include    <dsmcc.h>

DSM_u_longlong  DSM_Access__get_Size
                        ( DSM_Access            o,
                        CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object whose size is sought
ev	*(in/out)* CORBA environment

Returns

the size (in octets) of the object's attributes

Exceptions

(standard)

Description

Returns the size of an object's attributes.

Notes

1. This operation requires READER privilege.
2. Attributes also may be set using DSM::Directory::put, and read using DSM::Directory::get.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.2.

DSM::Base − common operations for deletion of objects and termination of access to objects

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_Base ;
```

Description

The DSM::Base object defines common operations for the deletion of objects and termination of access to objects.

Notes

1. A DSM::Base object is an abstract object, and is not instantiable. It is defined so that its data structures and attributes may be inherited by other objects.

2. The close operation terminates access to an object. The destroy operation deletes the object itself.

3. The DSM::Base interface maps directly to the remote (SI) interface. (Refer to the DSM−CC specification, Section 5.6.5.1.) This implies that DSM::Base operations are remote only, and do not necessarily imply any local operation.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.1.

DSM::Base::close − indicate object access no longer required

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Base_close_ACR              DSM_READER

DSM_RequestHandle       DSM_Base_close
                                ( DSM_Base              o,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object to which access is no longer required
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

(standard)

Description

Indicate that access to an object is no longer required. Resources needed to communciate with the object o are deleted. The object itself is not deleted, and may be accessed again after obtaining a new reference to the object.

Notes

1. If a parent DSM::Composite object is closed, its child objects are also closed as a result of this operation.

2. Object references are resources. A system may have a limit to the maximum number of open object reference. An attempt to surpass the limit will result in a DSM::OPEN_LIMIT exception. This operation may be used to reduce the number of open references, thus avoiding the exception.

3. This operation requires READER privilege.

4. Because the DSM::Base interface maps directly to the remote (SII) interface, this operation does not necessarily affect local data or resources. In particular, it is still important to free local object references by calling CORBA::Object::release for each such reference. In some implementations, the object references are kept in user address space, and the CORBA implementation cannot do garbage collection efficiently.

5. The DSM−CC specification says this call will delete the object reference, which is usually taken to be a local operation. In contradiction, the DSM−CC specification also says that this call is mapped directly to the remote (SI) interface, which makes it a server operation. Object references in C are implemented as pointers to `void`, and in many implementations multiple references to the same object are simply pointers with the same value, since they all share a common data structure. In more complex environments, however, especially when memory mapping is used, not all object references point to the same data structures. The CORBA specification anticipates the general case, while the DSM−CC specification seems to have some specific and implicit conception of the way the specification will be implemented. The DSM−CC conception cannot be divined completely from the specification, and is therefore ambiguous. Bionic Buffalo implementations give priority to CORBA over DSM−CC, as long as over−the−wire interoperability is not compromised. Therefore, this call is taken as an invocation of a remote operation, and will not to obviate the need for `CORBA::Object::release`.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.1.2.

DSM::Base::destroy − destroy an object instance

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Base_destroy_ACR            DSM_OWNER

DSM_RequestHandle       DSM_Base_destroy
                                ( DSM_Base              o,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object to be destroyed
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

(standard)

Description

Destroys an object, and releases the associated resources.

Notes

1. After this operation, the object no longer exists, and the object reference is no longer valid.

2. This operation requires OWNER privilege.

3. Please see the notes under DSM::Base::close regarding the need for CORBA::Object::release.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.1.3.

DSM::CompatibilityDescriptor − inventory of hardware and software

Synopsis − C

```
#include    <dsmcc.h>

typedef struct DSM_SubDescriptor
        {
        CORBA_octet                     subDescriptorType ;
        CORBA_octet                     subDescriptorLength ;
        CORBA_sequence_octet            additionalInformation ;
        }
        DSM_SubDescriptor ;

typedef struct DSM_sequence_SubDescriptor
        {
        CORBA_unsigned_long             _maximum ;
        CORBA_unsigned_long             _length ;
        DSM_SubDescriptor             * _buffer ;
        }
        DSM_sequence_SubDescriptor ;

typedef struct DSM_InterfaceDescriptor
        {
        CORBA_octet                     descriptorType ;
        CORBA_octet                     descriptorLength ;
        CORBA_unsigned_long             specifier ;
        CORBA_unsigned_short            model ;
        CORBA_unsigned_short            version ;
        DSM_sequence_SubDescriptor      subDescriptorList ;
        }
        DSM_InterfaceDescriptor ;

typedef struct DSM_sequence_InterfaceDescriptor
        {
        CORBA_unsigned_long             _maximum ;
        CORBA_unsigned_long             _length ;
        DSM_InterfaceDescriptor       * _buffer ;
        }
        DSM_sequence_InterfaceDescriptor ;

typedef struct DSM_CompatibilityDescriptor
        {
        CORBA_unsigned_short            length ;
        DSM_sequence_InterfaceDescriptor
                                        interfaceDescriptorList ;
        }
        DSM_CompatibilityDescriptor ;
```

Description

The DSM::CompatibilityDescriptor structure is used to convey an inventory of client hardware or software to a server. The server may use this information to make decisions about which objects and data might be appropriate to download to a client.

A CompatibilityDescriptor is a set of InterfaceDescriptors. Values for InterfaceDescriptors are defined by organizations ("specifiers"). Each specifier may also define SubDescriptors to convey additional information.

In the client User−User specification, DSM::CompatibilityDescriptor structures are used in the DSM::DownloadSI interface, which is not normally employed by applications. It is usually only the concern of system software.

Notes

1. The descriptorType distinguishes the type of hardware or software referenced by the descriptor. Allowable values are:

descriptor type value	***use***
0x00	pad descriptor (used to provide alignment of following data, as needed)
0x01	system hardware descriptor
0x02	system software descriptor
0x03..0x3f	reserved to ISO/IEC 13818−6
0x40..0xff	defined by implementor or user

2. The specifierType distinguishes the format of the specifierData.

specifier type value	***use***
0x00	reserved to ISO/IEC 13818−6
0x01	IEEE OUI (organization uniqe identifier) (described in IEEE 802.1990)
0x02..0x7f	reserved to ISO/IEC 13818−6
0x80..0xff	defined by implementor or user

3. The specifierData field contains a unique identifier for the specifying organization.

4. The model and version are defined by the specifying organization. Values of all ones imply "all models" or "all versions".

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Sections 5.6.6.5 and 6.1.

`DSM::Composite` − relate child objects to a common parent

documentation to be supplied in a future release

DSM::Config − control invocation behaviour

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_Config ;

typedef CORBA_unsigned_long             DSM_RequestHandle ;

/* DSM::Config::DeferredSync */

void            DSM_Config__set_DeferredSync
                        ( DSM_Config                    o,
                        CORBA_boolean                   DeferredSync,
                        CORBA_Environment             * ev ) ;
CORBA_boolean   DSM_Config__get_DeferredSync
                        ( DSM_Config                    o,
                        CORBA_Environment             * ev ) ;
```

Description

The DSM::Config object provides interfaces to allow control of invocation behaviour. By setting the value of the DeferredSync attribute, an application can change between synchronous and synchrouous deferred operation.

Two operations are provided: inquire, to determine the status of a deferred operation; and wait, to wait for a deferred operation to complete.

Notes

1. The DSM::Config interface is abstract, and is not instantiable. Each object which supports deferred synchronous behaviour inherits and supports this interface.

2. When DeferredSync = FALSE, invocation of operations returning DSM_RequestHandle will block until the server completes the operation and results have been received. The RequestHandle return from such operations will be zero, and can be ignored.

3. When DeferredSync = TRUE, invocation of operations returning DSM_RequestHandle will not block. Such invocations will return immediately with a non−zero value of RequestHandle. Meanwhile, the request will be sent to the server. The inquire and wait calls may be used later with the RequestHandle as a parameter to determine the status of the operation.

4. A deferred call to a remote object may fail locally. In such cases, the inquire operation will immediately return an exception. It is good practice to call inquire right after invoking the remote operation, to determine if any local exceptions have been raised.

5. There is a separate `DeferredSync` flag for each application thread. The initial value of every such flag is `FALSE`.

6. A thread may have more than one operation outstanding by use of this mechanism.

7. Timeout values may be set using the `ESP::operation::timeout` attribute.

8. In the Bionic Buffalo implementations, `RequestHandle`s are unique across all threads in a process. One thread may safely inquire or wait on a `RequestHandle` acquired by some different thread.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.8.

DSM::Config::inquire − check status of deferred operation

Synopsis − C

```
#include    <dsmcc.h>

void            DSM_Config_inquire
                        ( DSM_Config                    o,
                        DSM_RequestHandle               aRequest,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the Config object
aRequest	*(in)* the RequestHandle returned from the call
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

(standard exceptions, plus any exception which might have been returned by the original operation*)*

Description

Checks the status of an operation called with synchronous deferred behaviour.

Notes

1. If inquire returns without an exception, then the operation is completed. The return parameters from the original deferred call now will be valid. The RequestHandle is no longer valid, and may eventually be recycled by the DSM−CC library.

2. If inquire returns an exception with COMPLETED_NO or COMPLETED_MAYBE, then the operation has not completed. Other exceptions are returned by the deferred operation itself.

3. This operation requires no particular privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.8.3.

DSM::Config::wait − wait until a deferred operation has completed

Synopsis − C

```
#include    <dsmcc.h>

void            DSM_Config_wait
                        ( DSM_Config                    o,
                        DSM_RequestHandle               aRequest,
                        CORBA_Environment             * ev ) ;
```

Arguments

o	*(in)* the Config object
aRequest	*(in)* the RequestHandle returned from the call
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

(standard exceptions, plus any exception which might have been returned by the original operation*)*

Description

Wait for completion an operation called with synchronous deferred behaviour.

Notes

1. Blocks the current thread until the operation pertaining to aRequest has completed.
2. This operation requires no particular privilege.
3. This operation is similar to CORBA::Request::get_response.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control,* Sept 1st 1998, Section 5.5.2.8.4.

DSM::Directory − binding names to objects or data

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_Directory ;

typedef CORBA_char                      DSM_PathType ;

#define DSM_DEPTH                       'D'
#define DSM_BREADTH                     'B'

typedef struct DSM_Step
        {
        CosNaming_NameComponent         name ;
        CORBA_boolean                   process ;
        }
        DSM_Step ;

typedef struct DSM_PathSpec
        {
        CORBA_unsigned_long             _maximum ;
        CORBA_unsigned_long             _length ;
        DSM_Step                      * _buffer ;
        }
        DSM_PathSpec ;

DSM_PathSpec  * DSM_PathSpec__alloc ( void ) ;
DSM_Step      * DSM_PathSpec_allocbuf ( CORBA_unsigned_long len ) ;

typedef CORBA_sequence_any              DSM_PathValues ;
```

Description

The Directory interface provides mechanisms for binding names to objects or data. It inherits and extends the mechanisms of CosNaming.

DSM::Directory inherits CosNaming::NameContext, and adds four new operations: open, close, get, and put. The close operation is equivalent to DSM::Base::close. The open, get, and put operations manipulate a new data type, the PathSpec.

A DSM::Step is an extension of a CosNaming::NameComponent. It adds an additional boolean element, process. A PathSpec is similar to a CosNaming::Name, except that it is composed of Steps rather than NameComponents. This means that each element of the sequence adds a binary process flag.

When `CosNaming` operations resolve a `Name`, each `NamingElement` is resolved to a `NamingContext`, except the last which may instead be resolved to any arbitrary object. Each successive `NamingElement` represents an object contained within the previous `NamingContext`. The result is a single object reference, bound to the last name in the path. The path assumes increasing "depth" at each element.

The `DSM::Directory` operations `open`, `get`, and `put`, differ from a `CosNaming::NamingContext::resolve` in three ways:

1. An additional parameter, `PathType`, specifies whether the `PathSpec` is to be traversed by depth (as with `CosNaming` paths), or by breadth. A `PathSpec` taken by breadth is a list of bindings in a single context.

2. The `Directory::open` operation resolves names to objects, just as does `NamingContext::resolve`. The `Directory::put` and `Directory::get` operations, on the other hand, resolve names to data values.

3. The `process` flag at each `Step` is used to specify whether the name at that `Step` is to be resolved. Instead of representing a single value, a `PathSpec` represents as many values as it has `process` flags set.

The other `Directory` operations, inherited from `DSM::Access` and `CosNaming::NamingContext`, behave as they do in their base definitions, except that appropriate user privilege is required for successful invocation.

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.

DSM::Directory::bind − bind object reference to name

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_bind_ACR          DSM_WRITER

DSM_RequestHandle       DSM_Directory_bind
                                ( CosNaming_NamingContext       o,
                                CosNaming_Name                * n,
                                CORBA_Object                  * obj,
                                CORBA_Environment             * ev ) ;
```

Description

Binds a name to an object within a naming context. Inherited from, and the same as, `CosNaming::NamingContext::bind` (*q.v.)*, except for the return value.

Notes

1. Requires `WRITER` privilege.
2. The `RequestHandle` return value differs from that of `CosNaming::NamingContext::bind`, in order to implement deferred synchronous requests. (See `DSM::Config` for more information.)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.5.

DSM::Directory::bind_context − bind directory to name

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_bind_context_ACR          DSM_WRITER

DSM_RequestHandle       DSM_Directory_bind_context
                                ( CosNaming_NamingContext       o,
                                CosNaming_Name                * n,
                                CosNaming_NamingContext         nc,
                                CORBA_Environment             * ev ) ;
```

Description

Binds a name to an object within a naming context. Inherited from, and the same as, `CosNaming::NamingContext::bind_context` (*q.v.)*, except for the return value.

Notes

1. Requires `WRITER` privilege.
2. The `RequestHandle` return value differs from that of `CosNaming::NamingContext::bind_context`, in order to implement deferred synchronous requests. (See `DSM::Config` for more information.)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.6.

DSM::Directory::bind_new_context − create a new Directory and bind it to a name

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_bind_new_context_ACR      DSM_OWNER

#define DSM_Directory_bind_new_context          \
                        CosNaming_NamingContext_bind_new_context
```

Description

Create a new Directory and bind its name. Inherited from, and the same as, CosNaming::NamingContext::bind_new_context (*q.v.)*.

Notes

1. Requires OWNER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.11.

DSM::Directory::close − indicate directory access no longer required

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_close_ACR                 DSM_READER

DSM_RequestHandle       DSM_Directory_close
                                ( DSM_Directory         o,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the directory to be closed
ev	*(in/out)* CORBA environment

Returns

DSM_ReqestHandle

Exceptions

(standard)

Description

Indicate that access to a directory is no longer required.

Notes

1. Object references are resources. A system may have a limit to the maximum number of open object references. (An attempt to surpass the limit will result in a DSM::OPEN_LIMIT exception.) This operation may be used to reduce the number of open references, thus avoiding the exception. Its use is prudent in standard practice, but not required.

2. This operation requires READER privilege.

3. Because the DSM::Base interface maps directly to the remote (SII) interface, this operation does not necessarily affect local data or resources. In particular, it is still important to free local object references by calling CORBA::Object::release for each such reference. In some implementations, the object references are kept in user address space, and the CORBA implementation cannot do garbage collection efficiently.

4. The DSM−CC specification says this call will delete the object reference, which is usually taken to be a local operation. In contradiction, the DSM−CC specification also says that this call is mapped directly to the remote (SI) interface, which makes it a server operation. Object references in C are implemented as pointers to `void`, and in many implementations multiple references to the same object are simply pointers with the same value, since they all share a common data structure. In more complex environments, however, especially when memory mapping is used, not all object references point to the same data structures. The CORBA specification anticipates the general case, while the DSM−CC specification seems to have some specific and implicit conception of the way the specification will be implemented. The DSM−CC conception cannot be divined completely from the specification, and is therefore ambiguous. Bionic Buffalo implementations give priority to CORBA over DSM−CC, as long as over−the−wire interoperability is not compromised. Therefore, this call is taken as an invocation of a remote operation, and will not to obviate the need for `CORBA::Object::release`.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.14.

DSM::Directory::destroy − destroy a directory

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_destroy_ACR               DSM_OWNER

DSM_RequestHandle       DSM_Directory_destroy
                                ( CosNaming_NamingContext       o,
                                CORBA_Environment             * ev ) ;
```

Description

Destroys a Directory object. Inherited from, and the same as, CosNaming::NamingContext::destroy (*q.v.*), except for the return value.

Notes

1. Requires OWNER privilege.

2. The RequestHandle return value differs from that of CosNaming::NamingContext::destroy, in order to implement deferred synchronous requests. (See DSM::Config for more information.)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.12.

DSM::Directory::get − return attribute values bound to a path specification

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_get_ACR                   DSM_READER

DSM_RequestHandle       DSM_Directory_get
                                ( DSM_Directory         o,
                                DSM_PathType            aPathType,
                                DSM_PathSpec          * rPathSpec,
                                DSM_PathValues       ** rPathValues,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the directory from which attribute values are sought
aPathType	*(in)* the manner of path traversal
aPathSpec	*(in)* the path specification
rPathValues	*(out)* attribute values returned
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::NO_AUTH
DSM::UNK_USER
DSM::SERVICE_XFR
CosNaming::NamingContext::NOT_FOUND
CosNaming::NamingContext::CANNOT_PROCEED
CosNaming::NamingContext::INV_NAME

Description

Returns attribute values corresponding to the given path specification.

Notes

1. Requires READER privilege.

2. If aPathType is DSM_DEPTH, then the steps of aPathSpec correspond to <*directory*>, <*directory*>, ..., <*object*>, <*attribute*>. The last step names the attribute. Only a single attribute value is returned.

3. If aPathType is DSM_BREADTH, then the steps of aPathSpec correspond to <*object*>, <*attribute*>, <*attribute*>, ..., <*attribute*>. Multiple attribute values may be returned, depending on the values of process in the attribute steps.

4. Each path node is examined sequentially, but not atomically. Other operations may occur between processing individual nodes.

5. If the resolution of any specific node fails, then the entire operation returns the appropriate exception.

6. The kind of attribute (in each path specification step) does not need to be specified. (That is, the value may be `NULL`.) The identifier is the name of the attribute.

7. The data structures for the `rPathValues` are determined by their type, and may be identified by the `TypeCode` of the `CORBA_any` structure in each element.

8. This operation is use to determine attribute values, not object references. To resolve object references, use `Directory::resolve` or `Directory::open`.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.15.

DSM::Directory::Hist − version and time of a directory

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_Hist_get_ACR      DSM_Access_Hist_get_ACR
#define DSM_Directory_Hist_put_ACR      DSM_Access_Hist_put_ACR

#define DSM_Directory__get_Hist         DSM_Access__get_Hist
#define DSM_Directory__set_Hist         DSM_Access__set_Hist
```

Description

Returns or sets the version and time of a directory. Inherited from, and the same as, `DSM::Acess::Hist` (*q.v.*).

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.2.

DSM::Directory::list − return list of bindings

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_list_ACR                  DSM_READER

DSM_RequestHandle       DSM_Directory_list
                                ( CosNaming_NamingContext       o,
                                CORBA_unsigned_long     how_many,
                                CosNaming_BindingList        ** bl,
                                CosNaming_BindingIterator     * bi,
                                CORBA_Environment             * ev ) ;
```

Description

Returns a list of bindings from a directory. This operation is inherited from, and the same as, `CosNaming::NamingContext::list` (*q.v.*), except for the return value.

Notes

1. Requires `READER` privilege.

2. The `RequestHandle` return value differs from that of `CosNaming::NamingContext::list`, in order to implement deferred synchronous requests. (See `DSM::Config` for more information.)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.3.

DSM::Directory::Lock − status of directory read and write locks

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_Lock_get_ACR      DSM_Access_Lock_get_ACR
#define DSM_Directory_Lock_put_ACR      DSM_Access_Lock_put_ACR

#define DSM_Directory__get_Lock         DSM_Access__get_Lock
#define DSM_Directory__set_Lock         DSM_Access__set_Lock
```

Description

Returns or sets the lock attribute of a directory. Inherited from, and the same as `DSM::Access::Lock` (*q.v.*).

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.2.

DSM::Directory::new_context − create a new directory

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_new_context_ACR           DSM_OWNER

#define DSM_Directory_new_context               \
                                    CosNaming_NamingContext_new_context
```

Description

Creates a new directory on the same server as an existing directory. Inherited from, and the same as, `CosNaming::NamingContext::new_context` (*q.v.*).

Notes

1. Requires `OWNER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.10

DSM::Directory::open − resolve the objects of a path

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_open_ACR                  DSM_READER

DSM_RequestHandle       DSM_Directory_open
                                ( DSM_Directory         o,
                                DSM_PathType            aPathType,
                                DSM_PathSpec          * rPathSpec,
                                DSM_ObjRefs          ** resolvedRefs,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the directory to be used as context for the path specification
aPathType	*(in)* the method of path traversal
aPathSpec	*(in)* the path to be resolved
resolvedRefs	*(out)* the resolved objects from the path
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::NO_AUTH
DSM::UNK_USER
DSM::SERVICE_XFR
CosNaming::NamingContext::NOT_FOUND
CosNaming::NamingContext::CANNOT_PROCEED
CosNaming::NamingContext::INV_NAME
DSM::OPEN_LIMIT

Description

Finds the objects associated with the names in a given path.

Notes

1. Requires READER privilege.

2. If aPathType is DSM_DEPTH, then the steps of aPathSpec correspond to <*directory*>, <*directory*>, ..., <*directory*>, <*object or directory*>. Each node for which process = 1 is resolved to an object reference in the output list.

3. If `aPathType` is `DSM_BREADTH`, then the steps of `aPathSpec` correspond to objects within the naming context. The objects (which may be directory objects) are all within the given directory, and each (for which `process = 1`) is resolved to an object reference for the output list.

4. Each path node is examined sequentially, but not atomically. Other operations may occur between processing individual nodes.

5. If the resolution of any specific node fails, then the entire operation returns the appropriate exception.

6. The function of this operation is similar to that of `resolve`. However, `open` allows multiple simultaneous resolutions of names.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.13

`DSM::Directory::Perms` − access permission information for a directory

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_Perms_get_ACR     DSM_Access_Perms_get_ACR
#define DSM_Directory_Perms_put_ACR     DSM_Access_Perms_put_ACR

#define DSM_Directory__get_Perms        DSM_Access__get_Perms
#define DSM_Directory__set_Perms        DSM_Access__set_Perms
```

Description

Returns or sets the permission attribute of a directory. Inherited from, and the same as `DSM::Access::Perms` (*q.v.*).

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.2.

DSM::Directory::put − bind attribute values to a path specification

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_put_ACR                   DSM_WRITER

DSM_RequestHandle       DSM_Directory_put
                                ( DSM_Directory         o,
                                DSM_PathType            aPathType,
                                DSM_PathSpec          * rPathSpec,
                                DSM_PathValues        * rPathValues,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the directory to which attribute values are to be bound
aPathType	*(in)* the manner of path traversal
aPathSpec	*(in)* the path specification
rPathValues	*(in)* attribute values
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::NO_AUTH
DSM::UNK_USER
DSM::SERVICE_XFR
CosNaming::NamingContext::NOT_FOUND
CosNaming::NamingContext::CANNOT_PROCEED
CosNaming::NamingContext::INV_NAME

Description

Sets attribute values corresponding to the given path specification.

Notes

1. Requires WRITER privilege.

2. If aPathType is DSM_DEPTH, then the steps of aPathSpec correspond to <*directory*>, <*directory*>, ..., <*object*>, <*attribute*>. The last step names the attribute. Only a single attribute value is bound.

3. If aPathType is DSM_BREADTH, then the steps of aPathSpec correspond to <*object*>, <*attribute*>, <*attribute*>, ..., <*attribute*>. Multiple attribute values may be bound, depending on the values of process in the attribute steps.

4. Each path node is examined sequentially, but not atomically. Other operations may occur between processing individual nodes.

5. If the resolution of any specific node fails, then the entire operation returns the appropriate exception.

6. The kind of attribute (in each path specification step) does not need to be specified. (That is, the value may be `NULL`.) The identifier is the name of the attribute.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.16.

DSM::Directory::rebind − rename an object

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_rebind_ACR                DSM_WRITER

DSM_RequestHandle       DSM_Directory_rebind
                                ( CosNaming_NamingContext       o,
                                CosNaming_Name                * n,
                                CORBA_Object                  * obj,
                                CORBA_Environment             * ev ) ;
```

Description

Renames an object within a given context. This operation replaces an existing name binding in the context. Inherited from, and the same as, `CosNaming::NamingContext::rebind` (*q.v.*), except for the return value.

Notes

1. Requires `WRITER` privilege.

2. The `RequestHandle` return value differs from that of `CosNaming::NamingContext::rebind`, in order to implement deferred synchronous requests. (See `DSM::Config` for more information.)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.7.

DSM::Directory::rebind_context − rename a directory

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_rebind_context_ACR        DSM_WRITER

DSM_RequestHandle       DSM_Directory_rebind_context
                                ( CosNaming_NamingContext       o,
                                CosNaming_Name                * n,
                                CosNaming_NamingContext         nc,
                                CORBA_Environment             * ev ) ;
```

Description

Binds a new name to an object within a naming context. Inherited from, and the same as, `CosNaming::NamingContext::rebind_context` (*q.v.*), except for the return value.

Notes

1. Requires `WRITER` privilege.
2. The `RequestHandle` return value differs from that of `CosNaming::NamingContext::rebind_context`, in order to implement deferred synchronous requests. (See `DSM::Config` for more information.)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.8.

`DSM::Directory::resolve` − return object reference for given name

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_resolve_ACR               DSM_READER

#define DSM_Directory_resolve   CosNaming_NamingContext_resolve
```

Description

Given a name, returns an object reference for an object. Inherited from, and the same as, `CosNaming::NamingContext::resolve` (*q.v.*).

Notes

1. Requires `READER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.4.

DSM::Directory::Size − size of a directory

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_Size_get_ACR      DSM_Access_Size_get_ACR

#define DSM_Directory__get_Size         DSM_Access__get_Size
```

Description

Returns the size of an directory's attributes. Inherited from, and the same as, `DSM::Access:Size` (*q.v.*).

Notes

1. This operation requires `READER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.2.

DSM::Directory::unbind − remove name from an object

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Directory_unbind_ACR                DSM_WRITER

DSM_RequestHandle       DSM_Directory_unbind
                                ( CosNaming_NamingContext       o,
                                CosNaming_Name                * n,
                                CORBA_Environment             * ev ) ;
```

Description

Remove the binding for a name within a context. Inherited from, and the same as, `CosNaming::NamingContext::unbind` (*q.v.*), except for the return value.

Notes

1. Requires `WRITER` privilege.

2. The `RequestHandle` return value differs from that of `CosNaming::NamingContext::unbind`, in order to implement deferred synchronous requests. (See `DSM::Config` for more information.)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.5.9.

DSM::Download − manage Download client

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_Download ;

typedef struct DSM_ModuleInfo
        {
        DSM_u_short                     moduleId ;
        CORBA_octet                     moduleVersion ;
        DSM_u_long                      moduleSize ;
        CORBA_sequence_octet            moduleInfoBytes ;
                                        /* max length 255 octets */
        }
        DSM_ModuleInfo ;

typedef struct DSM_ModuleInfoList
        {
        CORBA_unsigned_long             _maximum ;
        CORBA_unsigned_long             _length ;
        DSM_ModuleInfo                * _buffer ;
        }
        DSM_ModuleInfoList ;

DSM_ModuleInfoList    * DSM_ModuleInfoList__alloc ( void ) ;
DSM_ModuleInfo        * DSM_ModuleInfoList_allocbuf
                                        ( CORBA_unsigned_long   len ) ;

typedef CORBA_sequence_unsigned_short   DSM_ModuleList ;
```

Description

The DSM::Download interface provides operations to manage download operations on objects which are explicitly accessible using the User−Network Download protocol. The Download object is implemented on the Download client, not on the server. (Although the server object is designated as having a DSM::Download interface, the server object presents a different interface, DSM::DownloadSI.)

Four operations are defined: info (to obtain information about modules to be downloaded), alloc (to allocate memory buffers), start (to begin the download process), and cancel (to terminate a download in progress).

Notes

1. The download protocol does not always require use of DSM::Download and DSM::DownloadSI interfaces. Objects with other interfaces (such as DSM::File) may sometimes be downloaded transparently, so the user is not aware of the access protocol.

2. A `DSM::Download` object may be viewed as a collection of *modules*, each of which is divided into *blocks*. Each module is known by a number, its *module id*. The blocks of a module are generally expected to be loaded into contiguous memory.

3. Interpretation of the `moduleInfoBytes` member of `DSM::ModuleInfo` is application specific.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.1.

DSM::Download::alloc − allocate memory buffers for a download

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Download_info_ACR           DSM_READER

DSM_RequestHandle       DSM_Download_alloc
                                ( DSM_Download          o,
                                CORBA_unsigned_short    aModuleId,
                                esp_buffer              rWriteBuffer,
                                esp_buffer            * rReadBuffer,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object to be downloaded
aModuleId	*(in)* the module of the object to be downloaded
rWriteBuffer	*(in)* an application−provided buffer object
rReadBuffer	*(out)* a system−provided buffer object
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_MODULE_ID

Description

Specifies or allocates a buffer for a module to be downloaded.

If rWriteBuffer is not NIL, then the application assigns the buffer and rReadBuffer is returned NIL. If rWriteBuffer is NIL, then the system assigns a buffer which is returned in rReadBuffer.

Notes

1. Requires READER privilege.

2. The aModuleId is taken from a previous DSM::Download::info operation. This buffer allocation must be done separately for each module.

3. The DSM−CC specification IDL defines rWriteBuffer and rReadBuffer as untyped objects, then continues on to violate the C mapping specification by treating them as simple buffer addresses. By narrowing the object type to esp::buffer buffer objects, Bionic Buffalo deviates from the DSM−CC specification in order to comply with the CORBA specifications.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.1.4.

DSM::Download::cancel − cancel a download in progress

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Download_cancel_ACR         DSM_READER

DSM_RequestHandle       DSM_Download_cancel
                                ( DSM_Download          o,
                                DSM_ModuleList        * aModuleList,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object being downloaded
aModuleList	*(in)* the list of modules for which download is to be cancelled
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_MODULE_ID
DSM::TIMEOUT

Description

Cancels a download in progress for specific modules.

Notes

1. Requires READER privilege.
2. The moduleIds within aModuleList are taken from previous DSM::Download::info and DSM::Download::start operations.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.1.6.

DSM::Download::info − obtain information about prerequisite download modules

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Download_info_ACR           DSM_READER

DSM_RequestHandle       DSM_Download_info
                                ( DSM_Download          o,
                                DSM_ModuleInfoList   ** rModulesInfo,
                                CORBA_Environment     * ev );
```

Arguments

o	*(in)* the object to be downloaded
rModulesInfo	*(in)* information about the modules in o
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

(standard exceptions)

Description

Provides information about the modules of a download object.

Notes

1. The data of a module are organized into modules, which in turn are divided into blocks. An application may require some or all of the modules of a download object: this is an implementation−dependent decision.

2. The returned information may be used to select among the available modules for downloading, based on implementation−dependent criteria. Once the modules are selected, DSM::Download::start may be used to initiate the download operation.

3. Requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.1.3.

DSM::Download::start − start download and transfer modules to client

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Download_start_ACR          DSM_READER

DSM_RequestHandle       DSM_Download_start
                                ( DSM_Download          o,
                                DSM_ModuleList        * aModuleList,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object being downloaded
aModuleList	*(in)* the list of modules for which download is to be started
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_MODULE_ID
DSM::TIMEOUT
DSM::MPEG_DELIVERY

Description

Begins a download operation for the indicated modules.

Notes

1. Requires READER privilege.
2. The module_ids within aModuleList are taken from a previous DSM::Download::info operation.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.1.5.

DSM::DownloadSI − server interface for DSM::Download objects

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                        DSM_DownloadSI ;
```

Description

The DSM::DownloadSI interface is presented by the server for DSM::Download objects. It allows access to lower−layer User−Network download protocol functions.

This interface is not normally used by applications. It is used internally by the DSM−CC client software.

Notes

5. This interface implements Model 2 of User−Network download. All control messages are exchanged using this interface. The data messages are exchanged over a separate high−speed channel.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.5.

DSM::DownloadSI::cancel − cancel download in progress

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_DownloadSI_cancel_ACR       DSM_READER

typedef struct DSM_DownloadSI_CancelRequest
        {
        CORBA_unsigned_short            moduleId ;
        CORBA_unsigned_short            blockNumber ;
        CORBA_octet                     downloadCancelReason ;
        CORBA_char                      privateDataLen ;
        CORBA_sequence_octet            privateDataBytes ;
        }
        DSM_DownloadSI_CancelRequest ;

DSM_RequestHandle       DSM_DownloadSI_cancel
                                ( DSM_DownloadSI        o,
                                DSM_DownloadSI_CancelRequest
                                                        * cancelReq,
                                CORBA_Environment       * ev ) ;
```

Arguments

o	*(in)* the object being downloaded
cancelReq	*(in)* information about the cancellation
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_MODULE_ID
DSM::TIMEOUT

Description

Cancels a download operation.

Notes

1. Requires READER privilege.

2. The privateDataBytes allow an implementation−dependent mechanism for providing more information about the cancellation.

3. The downloadCancelReason is taken from the following values:

Reason	Value	Sender	Description
	0x00		ISO/IEC 13818−6 reserved
rsnScenarioTimeout	0x01	Server, Client	Timer tCDownloadScenario expired
rsnInsufMem	0x02	Server, Client	Insufficient memory
rsnAuthDenied	0x03	Server	Download authorization denied
rsnFatal	0x04	Server, Client	Fatal error
rsnInfoRequestError	0x05	Server	The Download Server cannot accommodate the requested maximum blockSize or bufferSize
rsnCompatError	0x06	Server	The Download Server cannot determine an appropriate image from the compatibilityDescriptor provided by the client
rsnUnreliableNetwork	0x07	Server	The Client indicated protocol settings for a reliable download but Download Server has determined that the level of service provided by the network layer is unreliable
rsnInvalidData	0x08	Client	The Client received data which did not match the description in the moduleInfoByte fields in the DownloadInfoResponse or DownloadInfoIndication message
rsnInvalidBlock	0x09	Client	The Client received a block with an invalid moduleId or blockNumber
rsnInvalidVersion	0x0a	Client	The Client received a block with an unexpected value of the moduleVersion field

Reason	Value	Sender	Description
rsnAbort	0x0b	Client	The Client aborts the download scenario in progress
rsnRetrans	0x0c	Server, Client	The sender has reached the maximum allowed number of retransmissions
rsnBadBlockSize	0x0d	Client	The Client cannot support the selected value for blockSize
rsnBadWindow	0x0e	Client	The Client cannot support the selected value of windowSize
rsnBadAckPeriod	0x0f	Client	The Client cannot support the selected value for ackPeriod
rsnBadWindowTimer	0x10	Client	The Client cannot support the selected value for tCDownloadWindow
rsnBadScenarioTimer	0x11	Client	The Client cannot support the selected value for tCDownloadScenario
rsnBadCapabilities	0x12	Client	The Client cannot parse the compatibilityDescriptor
rsnBadModuleTable	0x13	Client	The Client cannot parse the module table
	0x14-0xef		ISO/IEC 13818−6 reserved
	0xf0-0xff		Private use

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.5.

DSM::DownloadSI::deinstall − unbind download configuration from server

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_DownloadSI_deinstall_ACR    DSM_OWNER

DSM_RequestHandle       DSM_DownloadSI_deinstall
                                ( DSM_DownloadSI        o,
                                DSM_CompatibilityDescriptor
                                                        * compatInfo,
                                CORBA_Environment       * ev ) ;
```

Arguments

o	*(in)* the download object bound to the configuration
compatInfo	*(in)* the download configuration to be unbound
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_COMPAT_INFO

Description

Unbinds a download configuration from the server. The compatibility descriptor specifies which configuration is to be unbound.

Notes

1. Requires OWNER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.5.

DSM::DownloadSI::info − obtain download information and negotiate parameters

Synopsis − C

```
#include    <dsmcc.h>

typedef struct DSM_InfoRequest
        {
        CORBA_unsigned_long             bufferSize ;
        CORBA_unsigned_short            maximumBlockSize ;
        DSM_CompatibilityDescriptor     userCompatibilitiesBytes ;
        CORBA_sequence_octet            privateDataBytes ;
        }
        DSM_InfoRequest ;

typedef struct DSM_InfoResponse
        {
        CORBA_unsigned_long             downloadId ;
        CORBA_unsigned_short            blockSize ;
        CORBA_octet                     windowSize ;
        CORBA_octet                     ackPeriod ;
        CORBA_unsigned_long             tCDownloadWindow ;
        CORBA_unsigned_long             tCDownloadScenario ;
        DSM_CompatibilityDescriptor     userCompatibilitiesBytes ;
        DSM_ModuleInfoList              modulesInfo ;
        CORBA_sequence_octet            privateDataBytes ;
        }
        DSM_InfoResponse ;

#define DSM_DownloadSI_info_ACR         DSM_READER

DSM_RequestHandle       DSM_DownloadSI_info
                                ( DSM_DownloadSI        o,
                                DSM_DownloadSI_InfoRequest
                                                * reqNegotiation,
                                DSM_DownloadSI_InfoResponse
                                              ** respNegotiation,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the download object
reqNegotiation	*(in)* the requested parameters
respNegotiation	*(out)* the reply parameters
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_COMPAT_INFO

`DSM::BUF_SIZE`
`DSM::BLOCK_SIZE`

Description

Obtain module information and negotiate flow−control parameters for a download operation.

Notes

1. Requires `READER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.5.

DSM::DownloadSI::install − bind download configuration to server

Synopsis − C

```
#include    <dsmcc.h>

typedef struct DSM_DownloadSI_ModuleInstallInfo
        {
        CORBA_unsigned_short            aModuleId ;
        CosNaming_Name                  n ;
        }
        DSM_DownloadSI_ModuleInstallInfo ;

typedef struct DSM_DownloadSI_sequence_ModuleInstallInfo
        {
        CORBA_unsigned_long             _maximum ;
        CORBA_unsigned_long             _length ;
        DSM_ModuleInstallInfo   *       _buffer ;
        }
        DSM_DownloadSI_sequence_ModuleInstallInfo ;

typedef DSM_DownloadSI_sequence_ModuleInstallInfo
                                DSM_DownloadSI_ModuleInstallList ;

#define DSM_DownloadSI_install_ACR      DSM_OWNER

DSM_RequestHandle       DSM_DownloadSI_install
                                ( DSM_DownloadSI        o,
                                DSM_CompatibilityDescriptor
                                                        * compatInfo,
                                DSM_DownloadSI_ModuleInfoList
                                                        * modulesInfo,
                                DSM_DownloadSI_ModuleInstallList
                                                        * pathsInfo,
                                CORBA_Environment       * ev ) ;
```

Arguments

o	*(in)* the download object to bind to the configuration
compatInfo	*(in)* the download configuration to be bound
modulesInfo	*(in)* information about the modules
pathsInfo	*(in)* paths to objects at server containing download module data
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_COMPAT_INFO
DSM::BAD_MODULE_INFO
DSM::INV_NAME

`DSM::NOT_FOUND`

Description

Binds a download configuration to a server.

Notes

1. Requires `OWNER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.5.

DSM::DownloadSI::proceed − transfer download data blocks

Synopsis − C

```
#include    <dsmcc.h>

typedef struct DSM_DownloadSI_DataRequest
        {
        CORBA_unsigned_short            moduleId ;
        CORBA_unsigned_short            blockNumber ;
        CORBA_octet                     downloadReason ;
        }
        DSM_DownloadSI_DataRequest ;

#define DSM_DownloadSI_proceed_ACR      DSM_READER

DSM_RequestHandle       DSM_DownloadSI_proceed
                                ( DSM_DownloadSI        o,
                                DSM_DownloadSI_DataRequest
                                                        * ackNack,
                                CORBA_Environment       * ev ) ;
```

Arguments

o	*(in)* the download object
ackNack	*(in)* the module and block to be downloaded
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_MODULE_ID
DSM::TIMEOUT
DSM::MPEG_DELIVERY

Description

Transfer a series of DownloadDataBlocks to the Client.

Notes

1. Requires READER privilege.

2. In DSM::DownloadSI::DataRequest, the moduleId and blockNumber identify the next block to be transferred. The total number of blocks to transfer is determined by the negotiated window size.

3. The value of downloadReason is taken from the following:

downloadReason	Value	Description
	0x00	ISO/IEC 13818−6 reserved
rsnStart	0x01	start request for download
rsnAckCont	0x02	acknowledge reception of previous blocks, and request continuation of transmission from indicated block
rsnNakRetransBlock	0x03	request retransmission of blocks starting from indicated block because blocks are missing
rsnNakRetransWindow	0x04	request retransmission of blocks starting from indicated block because timer tCDownloadWindow expired
rsnEnd	0x05	end download because all blocks are successfully received
	0x06-0xef	ISO/IEC 13818−6 reserved
	0xf0-0xff	private use

4. The download data is normally transferred over the channel identified by the DOWNLOAD_DATA_DOWN_USE tap.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.5.

DSM::Event − receive events in stream data

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                        DSM_Event ;
```

Description

The DSM::Event interface allows clients to subscribe to and receive events transmitted within DSM::Stream data.

The DSM::Event interface provides an attribute (DSM::Event::EventList), and three operations (subscribe, unsubscribe, and notify).

Notes

1. The DSM::Event interface is abstract, and is not instantiable. Each DSM::Stream object supporting these features inherits the DSM::Event interface and supports the operations.

2. The client first learns which events are available by examining the EventList. Events of interest are then subscribed.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.2.

DSM::Event::EventList − list of available events

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_sequence_char             DSM_eventName ;
                                        /* max 255 chars */
typedef struct DSM_EventList_T
        {
        CORBA_unsigned_long             _maximum ;
        CORBA_unsigned_long             _length ;       /* max 65535 */
        DSM_eventName                 * _buffer ;
        }
        DSM_EventList_T ;

#define DSM_Event_EventList_get_ACR     DSM_READER
#define DSM_Event_EventList_put_ACR     DSM_OWNER

DSM_EventList_T       * DSM_Event__get_EventList
                                ( DSM_Event             o,
                                CORBA_Environment     * ev ) ;
void                    DSM_Event__put_EventList
                                ( DSM_Event             o,
                                DSM_EventList_T       * EventList,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the event object
EventList	*(in)* list of events
ev	*(in/out)* CORBA environment

Returns

DSM_Event__get_EventList returns DSM_EventList
DSM_Event__put_EventList returns void

Exceptions

(standard exceptions)

Description

Sets or learns the list of available events.

Notes

1. DSM_Event__set_EventList requires OWNER privilege. DSM_Event__get_EventList requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.2

DSM::Event::notify − obtain event data from stream event descriptor

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Event_notify_ACR            DSM_READER

DSM_RequestHandle       DSM_Event_notify
                                ( DSM_Event             o,
                                DSM_StreamEvent      ** rStreamEvent,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the event object
rStreamEvent	*(out)* the event which arrived
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

(standard exceptions)

Description

Wait for an event to arrive.

Notes

1. Requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.2.5.

DSM::Event::subscribe − subscribe to receive an MPEG stream event

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Event_subscribe_ACR         DSM_READER

DSM_RequestHandle       DSM_Event_subscribe
                                ( DSM_Event             o,
                                CORBA_string            aEventName,
                                CORBA_unsigned_short  * eventId,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the event object
aEventName	*(in)* the event to subscribe
eventId	*(out)* the event identifier
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::INV_EVENT_NAME

Description

Request to be sent an event when it occurs.

Notes

1. Requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.2.3.

DSM::Event::unsubscribe − end subscription to an MPEG stream event

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Event_unsubscribe_ACR       DSM_READER

DSM_RequestHandle       DSM_Event_unsubscribe
                                ( DSM_Event             o,
                                CORBA_unsigned_short    eventId,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the event object
eventId	*(in)* the event identifier
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::INV_EVENT_ID

Description

Request no longer to receive an event.

Notes

1. Requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.2.4.

DSM::File − file system interface

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                        DSM_File ;
```

Description

The DSM::File interface allows access to conventional disk files.

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.

DSM::File::close − close a reference to a file

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_close_ACR              DSM_Base_close_ACR

#define DSM_File_close                  DSM_Base_close
```

Description

Indicate that access to a file is no longer required. Inherited from, and the same as, DSM::Base::close (*q.v.*).

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.

DSM::File::Content − contents of a file

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_Content_get_ACR        DSM_READER
#define DSM_File_Content_put_ACR        DSM_WRITER

DSM_opaque            * DSM_File__get_Content
                                ( DSM_File              o,
                                CORBA_Environment     * ev ) ;
void                    DSM_File__put_Content
                                ( DSM_File              o,
                                DSM_opaque            * Content,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the file object
Content	*(in)* file contents
ev	*(in/out)* CORBA environment

Returns

DSM_File__get_Content returns DSM_opaque
DSM_File__put_Content returns void

Exceptions

(standard exceptions)

Description

Sets or gets the contents of a file. This is equivalent to writing or reading the entire file at once.

Notes

1. DSM_File__set_Content requires READER privilege. DSM_File__get_Content requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.1.

DSM::File::ContentSize − size of file contents

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_ContentSize_get_ACR    DSM_READER

CORBA_unsigned_long_long        DSM_File__get_ContentSize
                                        ( DSM_File              o,
                                        CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the file object
ev	*(in/out)* CORBA environment

Returns

CORBA_unsigned_long_long − number of bytes in file

Exceptions

(standard exceptions)

Description

Determines the size of a file.

Notes

1. Requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.1.

`DSM::File::destroy` − destroy a file

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_destroy_ACR            DSM_Base_destroy_ACR

#define DSM_File_destroy                DSM_Base_destroy
```

Description

Destroys a file, and releases the associated resources. Inherited from, and the same as `DSM::Base::destroy` (*q.v.*).

Notes

1. This operation requires `OWNER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.2.

DSM::File::Hist − version and time of a file

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_Hist_get_ACR           DSM_Access_Hist_get_ACR
#define DSM_File_Hist_put_ACR           DSM_Access_Hist_put_ACR

#define DSM_File__get_Hist              DSM_Access__get_Hist
#define DSM_File__set_Hist              DSM_Access__set_Hist
```

Description

Returns or sets the version and time of a file. Inherited from, and the same as, DSM::Access::Hist (*q.v.*).

Notes

1. DSM_File__get_Hist requires READER privilege. DSM_File__set_Hist requires BROKER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.2.

DSM::File::Lock − status of file read and write locks

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_Lock_get_ACR           DSM_Access_Lock_get_ACR
#define DSM_File_Lock_put_ACR           DSM_Access_Lock_put_ACR

#define DSM_File__get_Lock              DSM_Access__get_Lock
#define DSM_File__set_Lock              DSM_Access__set_Lock
```

Description

Returns or sets the lock attribute of a file. Inherited from, and the same as, DSM::Access::Lock (*q.v.*).

Notes

1. DSM_File__get_Lock requires READER privilege. DSM_File__set_Lock requires WRITER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.2.

DSM::File::Perms − access permission information for a file

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_Perms_get_ACR          DSM_Access_Perms_get_ACR
#define DSM_File_Perms_put_ACR          DSM_Access_Perms_put_ACR

#define DSM_File__get_Perms             DSM_Access__get_Perms
#define DSM_File__set_Perms             DSM_Access__set_Perms
```

Description

Returns or sets the permission attribute of a file. Inherited from, and the same as, `DSM::Access::Perms` (*q.v.*).

Notes

1. Both of these operations require `OWNER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.2.

DSM::File::read − random access read from a file

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_read_ACR               DSM_READER

DSM_RequestHandle       DSM_File_read
                                ( DSM_File              o,
                                DSM_u_longlong          aOffset,
                                DSM_u_long              aSize,
                                CORBA_boolean           aReliable,
                                DSM_opaque           ** rData,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the file to be read
aOffset	*(in)* the offset in the file from which the read operation is to begin
aSize	*(in)* the number of octets to read from the file
aReliable	*(in)* indicates whether operation is to be re−tried in case of timeout or error
rData	*(out)* the data returned from the file
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::INV_OFFSET
DSM::INV_SIZE
DSM::READ_LOCKED

Description

Reads information from a file.

Notes

1. This operation requires READER privilege.

2. If the offset is within the boundaries of the file, then a length which would cause a read beyond the end of the file does not generate and exception. Trying to read beyond the end of the file will simply read to the end of the file.

3. DSM::INV_SIZE will be raised only if the capacity of the server, network, or client would be exceeded.

4. When there are multiple outstanding requests, they will be processed in the order received.

5. The first byte of the file has `aOffset = 0`.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.3.

`DSM::File::Size` − size of a file

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_Size_get_ACR           DSM_Access_Size_get_ACR

#define DSM_File__get_Size              DSM_Access__get_Size
```

Description

Returns the size of an file's attributes. Inherited from, and the same as, `DSM::Access::Size` (*q.v.*).

Notes

1. This operation requires `READER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.2.

DSM::File::write − random access write to a file

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_File_write_ACR              DSM_WRITER

DSM_RequestHandle       DSM_File_write
                                ( DSM_File              o,
                                DSM_u_longlong          aOffset,
                                DSM_u_long              aSize,
                                DSM_opaque            * rData,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the file to be written
aOffset	*(in)* the offset in the file at which the write operation is to begin
aSize	*(in)* the number of octets to write to the file
rData	*(out)* the data to be written to the file
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::INV_OFFSET
DSM::INV_SIZE
DSM::WRITE_LOCKED

Description

Writes information to a file.

Notes

1. This operation requires WRITER privilege.
2. DSM::INV_OFFSET is raised if aOffset > ContentSize.
3. DSM::INV_SIZE is raised if server, network, or client capacity would be exceeded.
4. Offsets begin at zero. (That is, the offset of the first byte is zero.)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.4.4.

`DSM::First` − obtain first objects

documentation to be supplied in a future release

`DSM::Interfaces` − interface repository operations

documentation to be supplied in a future release

DSM::Kind − determine supported interfaces

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_Kind ;

typedef DSM_u_long                      DSM_IFKind ;
typedef CORBA_sequence_unsigned_long    DSM_IFKindList ;

typedef struct DSM_IntfCode
        {
        DSM_u_long                      anIFKind ;
        CORBA_string                    kind ;
        CORBA_string                    repositoryId ;
        DSM_IFKindList                  includes ;
        }
        DSM_IntfCode ;
/*
 *      registered service tags (DSM_ik_*)
 */
#define DSM_ik_null             1230196480UL    /* 0x49534f00 */
#define DSM_ik_Base             1230196481UL    /* 0x49534f01 */
#define DSM_ik_Access           1230196482UL    /* 0x49534f02 */
#define DSM_ik_Stream           1230196483UL    /* 0x49534f03 */
#define DSM_ik_File             1230196484UL    /* 0x49534f04 */
#define DSM_ik_BindingIterator  1230196485UL    /* 0x49534f05 */
#define DSM_ik_NamingContext    1230196486UL    /* 0x49534f06 */
#define DSM_ik_Directory        1230196487UL    /* 0x49534f07 */
#define DSM_ik_Session          1230196488UL    /* 0x49534f08 */
#define DSM_ik_ServiceGateway   1230196489UL    /* 0x49534f09 */
#define DSM_ik_First            1230196490UL    /* 0x49534f0a */
#define DSM_ik_Download         1230196491UL    /* 0x49534f0b */
#define DSM_ik_Event            1230196492UL    /* 0x49534f0c */
#define DSM_ik_Composite        1230196493UL    /* 0x49534f0d */
#define DSM_ik_View             1230196494UL    /* 0x49534f0e */
#define DSM_ik_State            1230196495UL    /* 0x49534f0f */
#define DSM_ik_Interfaces       1230196496UL    /* 0x49534f10 */
#define DSM_ik_Security         1230196497UL    /* 0x49534f11 */
#define DSM_ik_Config           1230196498UL    /* 0x49534f12 */
#define DSM_ik_Lifecycle        1230196499UL    /* 0x49534f13 */
#define DSM_ik_Kind             1230196500UL    /* 0x49534f14 */
#define DSM_ik_SessionUU        1230196501UL    /* 0x49534f15 */
#define DSM_ik_SessionSI        1230196502UL    /* 0x49534f16 */
#define DSM_ik_DownloadSI       1230196503UL    /* 0x49534f17 */
```

Description

The DSM::Kind interface allows an application to determine what interfaces an object supports.

Notes

1. The `DSM::Kind` interface is abstract and not instantiable. It is defined so it might be inherited by other objects.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.2.10.

DSM::Kind::has_a − determine whether object supports an interface

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Kind_has_a_ACR              DSM_READER

DSM_RequestHandle       DSM_Kind_has_a
                                ( DSM_Kind              o,
                                DSM_IFKind              anIFKind,
                                CORBA_boolean         * aVerdict,
                                CORBA_Environment     * ev );
```

Arguments

o	*(in)* any object
anIFKind	*(in)* an interface kind
aVerdict	*(out)* whether object supports interface
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

(standard exceptions)

Description

Determines whether an object includes (inherits) a given interface.

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.10.2.

DSM::Kind::is_a − show interfaces supported by an object

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Kind_is_a_ACR               DSM_READER

DSM_RequestHandle       DSM_Kind_is_a
                                ( DSM_Kind              o,
                                DSM_IntfCode          ** whatItIs,
                                CORBA_Environment      * ev );
```

Arguments

o	*(in)* any object
whatItIs	*(out)* returned interface code for the object
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

(standard exceptions)

Description

Returns the interface code for an object.

Notes

1. This operation requires READER privilege.
2. Inherited interfaces are provided with the includes member of the interface code.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.10.3.

`DSM::LifeCycle` − create interoperable object reference

documentation to be supplied in a future release

DSM::Security − provide credentials

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_Security ;

typedef struct DSM_AuthRequest
        {
        CORBA_string                    aPassword ;
        DSM_opaque                      authData ;
        }
        DSM_AuthRequest ;

typedef DSM_AuthRequest                 DSM_AuthRequest_T ;
```

Description

The DSM::Security interface allows an application to provide credentials so access to an object might be granted.

The DSM::Perm attribute of an object includes a string (aPassword) and an opaque sequence of octets (authData). If either of these is not empty, then authentication is necessary to access that object.

An attempt to access a protected object will result in a DSM::NO_AUTH exception. The body of the exception data structure will carry a DSM::AuthRequest data structure. The contents of this data structure will be used by the application to prepare a similar DSM::AuthRequest structure passed to the authenticate operation.

Notes

1. The DSM::Security interface is abstract and not instantiable. It is defined so it might be inherited by other objects.

2. The DSM::NO_AUTH exception is raised by any operation attempting to resolve or use a protected object.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.7.

DSM::Security::authenticate − request authentication

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Security_authenticate_ACR   DSM_READER

DSM_RequestHandle       DSM_Security_authenticate
                                ( DSM_Security          o,
                                DSM_AuthRequest_T       authInfo,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the object requiring authentication
authInfo	*(in)* password or other security information
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

(standard exceptions)

Description

Requests authentication to an object.

Notes

1. This operation requires READER privilege.

2. The contents of authInfo are implementation specific, and are usually derived from the contents of the authInfo field of the preceeding DSM::NO_AUTH exception data structure. Common requirements include simple passwords, encrypted or hashed passwords, hashed PIN numbers, and digitally−signed responses to challenge data returned by the DSM::NO_AUTH exception.

3. The user or application is expected to know how to handle the DSM::NO_AUTH exception, and to know the appropriate response for the authenticate operation.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.7.1.

DSM::ServiceGateway − access to a service domain

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_ServiceGateway ;

#define DSM_ServiceGateway_bind_ACR             DSM_MANAGER
#define DSM_ServiceGateway_bind_context_ACR     DSM_MANAGER
#define DSM_ServiceGateway_rebind_ACR           DSM_MANAGER
#define DSM_ServiceGateway_rebind_context_ACR   DSM_MANAGER
#define DSM_ServiceGateway_unbind_ACR           DSM_MANAGER

/* additional inherited operations and attributes, listed below */
```

Description

A DSM::ServiceGateway is the "entrance" to a domain of services. Conceptually, it is similar to the top level directory of a server.

DSM::ServiceGateway inherits DSM::Directory and DSM::Session. The inherited Directory operations allow navigation within the domain represented by the ServiceGateway. The inherited Session operations allow attachment to additional ServiceGateways found relative to the current ServiceGateway.

Inherited Operations

All operations and attributes of DSM::ServiceGateway are inherited, directly or indirectly, from DSM::Directory, DSM::Session, DSM::Access and CosNaming::NamingContext.

IDL Operation or Attribute	***C procedure***	***Base Interface***
attach	attach	DSM::Session
bind	bind	CosNaming::NamingContext
bind_context	bind_context	CosNaming::NamingContext
bind_new_context	bind_new_context	CosNaming::NamingContext
close	close	DSM::Directory
destroy	destroy	CosNaming::NamingContext
detach	detach	DSM::Session
get	get	DSM::Directory
	_get_Hist	DSM::Access
	_get_Lock	DSM::Access
	_get_Perms	DSM::Access
	_get_Size	DSM::Access

IDL Operation or Attribute	*C procedure*	*Base Interface*
Hist		DSM::Access
list	list	CosNaming::NamingContext
Lock		DSM::Access
new_context	new_context	CosNaming::NamingContext
Perms		DSM::Access
put	put	DSM::Directory
rebind	rebind	CosNaming::NamingContext
rebind_context	rebind_context	CosNaming::NamingContext
resolve	resolve	CosNaming::NamingContext
	_set_Hist	DSM::Access
	_set_Lock	DSM::Access
	_set_Perms	DSM::Access
Size		DSM::Access
unbind	unbind	CosNaming::NamingContext

Notes

1. Access control for inherited operations is the same as for the base interface (DSM::Directory or DSM::Session), except for bind, bind_context, rebind, rebind_context, and unbind, which require MANAGER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.7.

DSM::ServiceGatewaySI − server interface for service gateway

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_ServiceGatewaySI ;

#define DSM_ServiceGatewaySI_bind_ACR           DSM_MANAGER
#define DSM_ServiceGatewaySI_bind_context_ACR   DSM_MANAGER
#define DSM_ServiceGatewaySI_rebind_ACR         DSM_MANAGER
#define DSM_ServiceGatewaySI_rebind_context_ACR DSM_MANAGER
#define DSM_ServiceGatewaySI_unbind_ACR         DSM_MANAGER

/* additional inherited operations and attributes, listed below */
```

Description

When the User−Network protocol is not used, then DSM::ServiceGatewaySI is the server (SII) interface implementing DSM::ServiceGateway operations.

DSM::ServiceGatewaySI inherits DSM::Directory and DSM::SessionSI. The inherited Directory operations allow navigation within the domain represented by the ServiceGateway. The inherited Session operations allow attachment to additional ServiceGateways found relative to the current ServiceGateway.

Inherited Operations

All operations and attributes of DSM::ServiceGatewaySI are inherited, directly or indirectly, from DSM::Directory, DSM::SessionSI, DSM::Access and CosNaming::NamingContext.

IDL Operation or Attribute	***C procedure***	***Base Interface***
attach	attach	DSM::SessionSI
bind	bind	CosNaming::NamingContext
bind_context	bind_context	CosNaming::NamingContext
bind_new_context	bind_new_context	CosNaming::NamingContext
close	close	DSM::Directory
destroy	destroy	CosNaming::NamingContext
detach	detach	DSM::SessionSI
get	get	DSM::Directory
	_get_Hist	DSM::Access
	_get_Lock	DSM::Access
	_get_Perms	DSM::Access
	_get_Size	DSM::Access

IDL Operation or Attribute	*C procedure*	*Base Interface*
Hist		DSM::Access
list	list	CosNaming::NamingContext
Lock		DSM::Access
new_context	new_context	CosNaming::NamingContext
Perms		DSM::Access
put	put	DSM::Directory
rebind	rebind	CosNaming::NamingContext
rebind_context	rebind_context	CosNaming::NamingContext
resolve	resolve	CosNaming::NamingContext
	_set_Hist	DSM::Access
	_set_Lock	DSM::Access
	_set_Perms	DSM::Access
Size		DSM::Access
unbind	unbind	CosNaming::NamingContext

Notes

1. Access control for inherited operations is the same as for the base interface (DSM::Directory or DSM::SessionSI), except for bind, bind_context, rebind, rebind_context, and unbind, which require MANAGER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.4.

DSM::ServiceGatewayUU − server interface for service gateway

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_ServiceGatewayUU ;

#define DSM_ServiceGatewayUU_bind_ACR           DSM_MANAGER
#define DSM_ServiceGatewayUU_bind_context_ACR   DSM_MANAGER
#define DSM_ServiceGatewayUU_rebind_ACR         DSM_MANAGER
#define DSM_ServiceGatewayUU_rebind_context_ACR DSM_MANAGER
#define DSM_ServiceGatewayUU_unbind_ACR         DSM_MANAGER

/* additional inherited operations and attributes, listed below */
```

Description

When the User−Network protocol is used, then DSM::ServiceGatewayUU is the server (SII) interface implementing those DSM::ServiceGateway operations inherited from DSM::Directory.

DSM::ServiceGatewayUU inherits DSM::Directory. The inherited Directory operations allow navigation within the domain represented by the ServiceGateway.

Inherited Operations

All operations and attributes of DSM::ServiceGatewayUU are inherited, directly or indirectly, from DSM::Directory, DSM::Access and CosNaming::NamingContext.

IDL Operation or Attribute	***C procedure***	***Base Interface***
bind	bind	CosNaming::NamingContext
bind_context	bind_context	CosNaming::NamingContext
bind_new_context	bind_new_context	CosNaming::NamingContext
close	close	DSM::Directory
destroy	destroy	CosNaming::NamingContext
get	get	DSM::Directory
	_get_Hist	DSM::Access
	_get_Lock	DSM::Access
	_get_Perms	DSM::Access
	_get_Size	DSM::Access
Hist		DSM::Access
list	list	CosNaming::NamingContext
Lock		DSM::Access
new_context	new_context	CosNaming::NamingContext

IDL Operation or Attribute	*C procedure*	*Base Interface*
Perms		DSM::Access
put	put	DSM::Directory
rebind	rebind	CosNaming::NamingContext
rebind_context	rebind_context	CosNaming::NamingContext
resolve	resolve	CosNaming::NamingContext
	_set_Hist	DSM::Access
	_set_Lock	DSM::Access
	_set_Perms	DSM::Access
Size		DSM::Access
unbind	unbind	CosNaming::NamingContext

Notes

1. Access control for inherited operations is the same as for the base interface (DSM::Directory), except for bind, bind_context, rebind, rebind_context, and unbind, which require MANAGER privilege.

2. When User−Network protocol is used, the DSM::ServiceGatewayUU interface implements the inherited DSM::Directory operations, while the DSM::Session operations inherited by DSM::ServiceGateway are implemented by the DSM::SessionUU interface.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.2.

DSM::Session − attach to or detach from a service gateway

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_Session ;
```

Description

The DSM::Session interface allows an application to establish or terminate a session with a DSM::ServiceGateway object. This is roughly equivalent to a "login" and "logout" operation to a server.

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.6.

DSM::Session::attach − attach to a service gateway domain

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_sequence_octet            DSM_ServiceDomain ;
typedef DSM_opaque                      DSM_UserContext ;
typedef CORBA_sequence_Object           DSM_ObjRefs ;

#define DSM_Session_attach_ACR          DSM_READER

DSM_RequestHandle       DSM_Session_attach
                                ( DSM_Session           o,
                                DSM_ServiceDomain     * aServiceDomain,
                                CosNaming_Name        * pathName,
                                DSM_UserContext       * savedContext,
                                DSM_ObjRefs          ** resolvedRefs,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the local session object
aServiceDomain	*(in)* server identifier (optional)
pathName	*(in)* path name to server (optional)
savedContext	*(in)* previous application context (optional)
resolvedRefs	*(out)* resolved object references (see Notes, below)
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::OPEN_LIMIT
DSM::NO_AUTH
DSM::UNK_USER
DSM::SERVICE_XFR
DSM::BAD_COMPAT_INFO
DSM::NO_RESUME
DSM::NOT_FOUND
DSM::CANNOT_PROCEED
DSM::INV_NAME

Description

Attach to a service gateway domain, establishing a session context.

Notes

1. Requires READER privilege.

2. Either `aServiceDomain` or `pathName` (or both) must be provided. If both are given, then `aServiceDomain` specifies the server, and `pathName` gives the path to a service gateway and (optionally) to a first service.

3. The `aServiceDomain` must be in NSAP address format. For an interactive session, it is the globally unique server network address. For a broadcast carousel session, it is the unique identifier of the carousel.

4. A previous session may be resumed by providing `savedContext` from a previous `detach` operation.

5. Depending on the `pathName`, this operation returns object references for a service gateway and (optionally) for a first service. If the first service is a composite object, then object references for the parent and child objects will be returned.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.6.3.

DSM::Session::detach − detach from a service gateway domain

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Session_detach_ACR          DSM_READER

DSM_RequestHandle       DSM_Session_detach
                                ( DSM_Session           o,
                                CORBA_boolean           aSuspend,
                                DSM_UserContext      ** savedContext,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the local session object
aSuspend	*(in)* whether or not to save the current application context
savedContext	*(out)* saved application context
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::NO_SUSPEND

Description

Detach from a service gateway domain, disconnecting from the gateway and from all objects of a session.

Notes

1. Requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.6.4.

DSM::SessionSI − server interface for session

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                        DSM_SessionSI ;
```

Description

The DSM::SessionSI interface implements DSM::Session on the server when the User−Network protocol is not used.

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.6.

DSM::SessionSI::attach − attach to a service gateway domain

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_SessionSI_attach_ACR        DSM_READER

void            DSM_SessionSI_attach
                        ( DSM_SessionSI         o,
                        DSM_InfoRequest       * downloadInfoReq,
                        CosNaming_Name        * pathName,
                        DSM_UserContext       * savedContext,
                        DSM_InfoResponse     ** downloadInfoResp,
                        DSM_ObjRefs          ** resolvedRefs,
                        CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the session object
downloadInfoReq	*(in)* download information request
pathName	*(in)* path name to server
savedContext	*(in)* previous application context (optional)
downloadInfoResp	*(out)* download information response
resolvedRefs	*(out)* resolved object references (see Notes, below)
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

DSM::OPEN_LIMIT
DSM::NO_AUTH
DSM::UNK_USER
DSM::SERVICE_XFR
DSM::BAD_COMPAT_INFO
DSM::NO_RESUME
DSM::NOT_FOUND
DSM::CANNOT_PROCEED
DSM::INV_NAME

Description

Attach to a service gateway domain, establishing a session context.

Notes

1. Requires READER privilege.

2. A previous session may be resumed by providing `savedContext` from a previous `detach` operation.

3. Depending on the `pathName`, this operation returns object references for a service gateway and (optionally) for a first service. If the first service is a composite object, then object references for the parent and child objects will be returned.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.3.

DSM::SessionSI::detach − detach from a service gateway domain

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_SessionSI_detach_ACR        DSM_READER

void            DSM_SessionSI_detach
                        ( DSM_SessionSI         o,
                        CORBA_boolean           aSuspend,
                        DSM_UserContext      ** savedContext,
                        CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the session object
aSuspend	*(in)* whether or not to save the current application context
savedContext	*(out)* saved application context
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

DSM::NO_SUSPEND

Description

Detach from a service gateway domain, disconnecting from the gateway and from all objects of a session.

Notes

1. Requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.3.

DSM::SessionUU − interface to U−N session protocol

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                        DSM_SessionUU ;
```

Description

The DSM::SessionUU interface is presented to the user−user layer by the client User−Network protocol implementation.

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.1.

DSM::SessionUU::attach − define uuData for session establishment

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_SessionUU_attach_ACR        DSM_READER

void            DSM_SessionUU_attach
                        ( DSM_SessionUU         o,
                        DSM_opaque             * downloadInfoReq,
                        CosNaming_Name         * pathName,
                        DSM_UserContext        * savedContext,
                        CORBA_Principal          aPrincipal,
                        IOP_ServiceContextList
                                               * inSC,
                        DSM_opaque            ** downloadInfoResp,
                        DSM_ConnBinder        ** downloadTaps,
                        DSM_ObjRefs           ** resolvedRefs,
                        IOP_ServiceContextList
                                              ** outSC,
                        CORBA_Environment      * ev ) ;
```

Arguments

o	*(in)* the session for which the uuData is to be defined
downloadInfoReq	*(in)* download info request
pathName	*(in)* path to desired service
savedContext	*(in)* previous application user context
aPrincipal	*(in)* identification of end user
inSC	*(in)* information for service context list
downloadInfoResp	*(out)* download info response
downloadTaps	*(out)* the download taps to be used for communication with the resolved service object
resolvedRefs	*(out)* objects resolved
outSC	*(out)* returned service context list information
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

DSM::NO_AUTH
DSM::BAD_COMPAT_INFO
DSM::UNK_USER
DSM::SERVICE_XFR
DSM::NO_RESUME
DSM::OPEN_LIMIT
DSM::NOT_FOUND
DSM::CANNOT_PROCEED

`DSM::INV_NAME`

Description

Use to specify the paramters to be included in the `uuData` fields of the User−Network Session Establishment messages. The input parameters are placed in the U−N `ClientSessionSetupRequest`, and the output parameters are taken from the U−N `ClientSessionSetupResponse`.

Notes

1. This procedure is used from within the User−User library, and is not normally used by an application. It is not available on all platforms, so it is not completely portable.

2. If the request is for a download session, then `downloadInfoReq` and `downloadInfoResp` are significant. (Otherwise, they are zero.)

3. The `pathName` has two steps: the path to an object of class `DSM::ServiceGatewayUU`, and (possibly) the name of a first service. A default service may be specified in the second step as a `NULL` string. The resolved object reference(s) are returned in `resolvedRefs`.

4. The taps for initial download of the resolved object are returned in `downloadTaps`, which is a sequence of `DSM::Tap`. If no initial download is required, then this sequence is empty.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.1.

DSM::SessionUU::detach − define uuData for session teardown

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_SessionUU_detach_ACR        DSM_READER

void            DSM_SessionUU_detach
                        ( DSM_SessionUU         o,
                        CORBA_boolean           aSuspend,
                        CORBA_Principal         aPrincipal,
                        IOP_ServiceContextList
                                                * inSC,
                        IOP_ServiceContextList
                                               ** outSC,
                        DSM_UserContext        ** savedContext,
                        CORBA_Environment       * ev ) ;
```

Arguments

o	*(in)* the session for which the uuData is to be defined
aSuspend	*(in)* whether or not to save the application context
aPrincipal	*(in)* the end user making the request
inSC	*(in)* information for service context list
outSC	*(out)* returned service context list information
savedContext	*(out)* the saved application context
ev	*(in/out)* CORBA environment

Returns

void

Exceptions

DSM::NO_SUSPEND

Description

Use to specify the paramters to be included in the uuData fields of the User−Network Session Teardown messages.

Notes

1. This procedure is used from within the User−User library, and is not normally used by an application. It is not available on all platforms, so it is not completely portable.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.6.6.1.

DSM::State − suspend or resume application state

Synopsis − C

```
#include    <dsmcc.h>

typedef CORBA_Object                    DSM_State ;
```

Description

The DSM::State interface allows an application to save and restore an object's state. It is used when activity is suspended for later resumption.

Notes

(none)

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.5.

DSM::State::resume − resume a service from a previous application state

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_State_resume_ACR            DSM_READER

DSM_RequestHandle       DSM_State_resume
                                ( DSM_State             o,
                                DSM_UserContext       * savedContext,
                                DSM_ObjRefs          ** restoredRefs,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* an object whose state was saved previously
savedContext	*(in)* the saved state
restoredRefs	*(out)* restored object references
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::NO_RESUME

Description

Restore an object's saved context.

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.5.3.

DSM::State::suspend − suspend application state for a service

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_State_suspend_ACR           DSM_READER

DSM_RequestHandle       DSM_State_suspend
                                ( DSM_State             o,
                                CORBA_boolean           aRelease,
                                DSM_UserContext      ** savedContext,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* an object whose state is to be saved
aRelease	*(in)* indication if resumption is expected soon
savedContext	*(out)* the saved state
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::NO_SUSPEND

Description

Save an object's state for resumption at a later time.

Notes

1. This operation requires READER privilege.

2. The aRelease parameter is a hint to the underlying transport for use in resource allocation.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.2.5.2.

`DSM::Stream` − the stream interface

An object with a `DSM::Stream` interface can transport audio, video, or other program content to a client. The operations emulate VCR−like controls for starting, stopping, fast−forward, and other control mechanisms.

The Stream State Machine: Simplified Version

The behaviour of a `DSM::Stream` object is described as a state machine. Transitions among states occur as a result of:

- conditions which arise during search and transport of the stream content
- operations invoked on the `DSM::Stream` object by a client.

The state variables are:

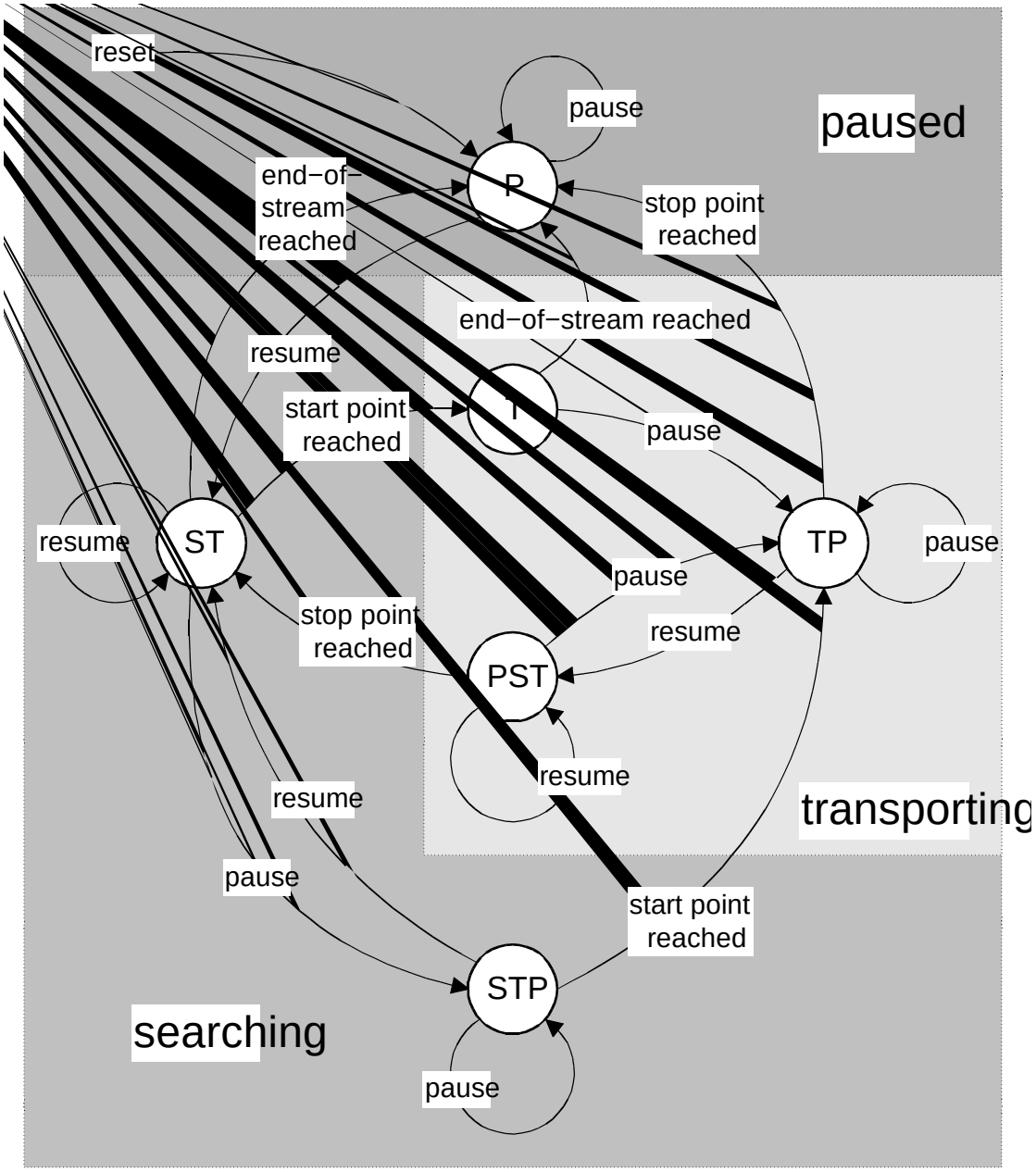
- the current state
- the current position within the stream
- the scale or rate of playback
- a "start" position within the stream (`startNPT`)
- a "stop" position within the stream (`stopNPT`)

The states themselves may be grouped into three categories, based on the transport activity which occurs during each state. The server is either *paused*, *transporting*, or *searching*, depending on the state.

Paused States

- OPEN (O). This is the initial state which exists when the `DSM::Stream` object is resolved. The stream is inactive. The current position is the beginning of the stream.
- PAUSE (P). During this state, the stream is inactive.
- END OF STREAM (EOS). This is the same as PAUSE, except that the current position is the end of the stream.

These three paused states undergo the same state transitions, so they are combined together in the simplified diagram below.



Transporting States

- TRANSPORT (T). The server is transporting, and will continue until end of stream is reached. No stop position within the stream is defined.

- TRANSPORT PAUSE (TP). A stop position is defined. The server will continue transporting until the stop position is reached.

- PAUSE SEARCH TRANSPORT (PST). Both start and stop positions are defined. The server will continue transporting until the stop position is reached, then it will enter the SEARCH TRANSPORT state to search for the start position.

Searching States

- SEARCH TRANSPORT (ST). The server is seeking to a defined start position. When it is reached, it will enter the TRANSPORT state and begin transporting until end of stream is reached.

- SEARCH TRANSPORT PAUSE (STP). Both start and stop positions are defined. The server seeks to the start position, then enters TRANSPORT PAUSE state to begin transporting until the stop position is reached.

The Stream State Machine: Detailed Version

The state transition diagram above is simplified by three factors:

1. the elimination of `play` and `jump` operations

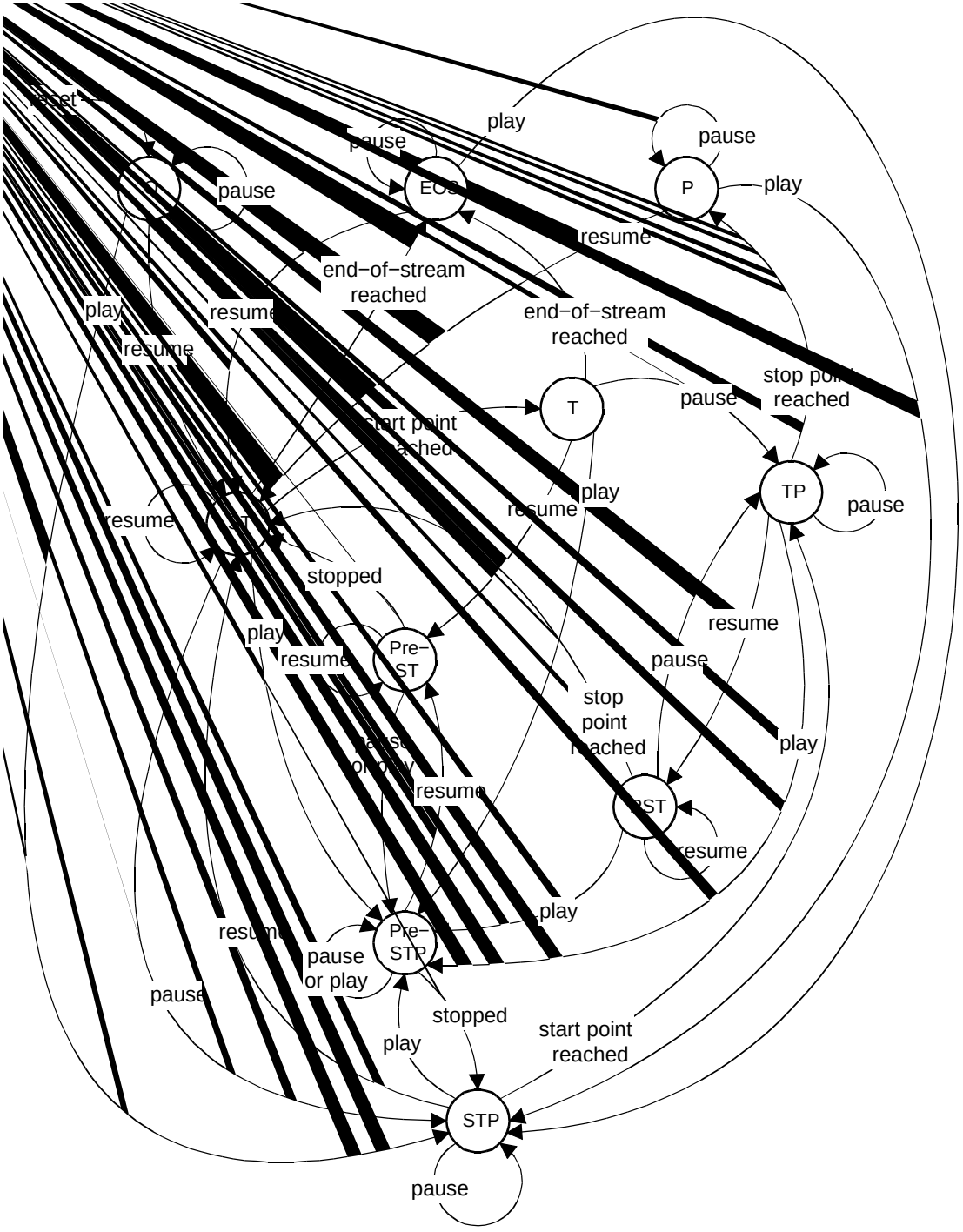
2. the combination of the three pause states (PAUSE, OPEN, and END OF STREAM) into a single PAUSE state

3. the elimination of two transient states, PRE−SEARCH TRANSPORT (PRE−ST) and PRE−SEARCH TRANSPORT PAUSE (PRE−STP)

The two transient states define the behaviour of the state machine when a transition is made from another state to SEARCH TRANSPORT (ST) or to SEARCH TRANSPORT PAUSE (STP) state. During such a transition, the server must stop the current search or transport operation before beginning the new search operation. The transient states (PRE−ST and PRE−STP) are defined as the conditions where the previous operation is being halted, but the next operation is not yet begun.

The following more complete state transition diagram for a `DSM::Stream` object includes the `play` operation. However, the `jump` operation, which is identical to `pause` followed by `resume`, is omitted for readability.

The following diagram also includes the two transient states, and separates the three pause states. Note that a `reset` operation can occur from any state, and will return the state machine to the OPEN state.



Normal Play Time and Rate of Play

The stream state machine addresses points in the stream using *Normal Play Time* (NPT) coordinates. NPT is expressed in seconds and microseconds, and is expressed independently of the underlying content frame rate or other factors. The NPT coordinate of the beginning of a stream is defined as zero.

A special value of NPT is 0x80000000, or "negative infinity". This represents "now" as perceived by the server. For example, if the server is told to resume play at 0x80000000, it will resume play at the current position.

While the stream is being transported, the rate of play is determined by the current value of the *scale* parameter. The scale is a fraction (numerator and denominator), and may be negative or positive. A magnitude greater than one indicates an accelerated rate of play, and a magnitude less than one indicates slow motion play.

Negative scale values are used to request reverse play.

Not all implementations will implement all possible rates of play, nor will they all be able to address the stream with the same temporal resolution. The server will use its best effort to satisfy the request.

DSM::Stream::close − close a reference to a stream

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_close_ACR            DSM_Base_close_ACR

#define DSM_Stream_close                DSM_Base_close
```

Description

Indicate that access to a stream is no longer required. Inherited from, and the same as, DSM::Base::close (*q.v.*).

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.

DSM::Stream::destroy − destroy a stream

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_destroy_ACR          DSM_Base_destroy_ACR

#define DSM_Stream_destroy              DSM_Base_destroy
```

Description

Destroys a stream, and releases the associated resources. Inherited from, and the same as, DSM::Base::destroy (*q.v.*).

Notes

1. This operation requires OWNER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.

DSM::Stream::Hist − version and time of a stream

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_Hist_T               DSM_Access_Hist_T
#define DSM_Stream_Hist_get_ACR         DSM_Access_Hist_get_ACR
#define DSM_Stream_Hist_put_ACR         DSM_Access_Hist_put_ACR

#define DSM_Stream__get_Hist            DSM_Access__get_Hist
#define DSM_Stream__set_Hist            DSM_Access__set_Hist
```

Description

Returns or sets the version and time of a stream. Inherited from, and the same as, `DSM::Access::Hist` (*q.v.*).

Notes

1. `DSM_Stream__get_Hist` requires `READER` privilege. `DSM_Stream__set_Hist` requires `BROKER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.

DSM::Stream::Info − stream identification and characteristics

Synopsis − C

```
#include    <dsmcc.h>

typedef struct DSM_Stream_Info_T
        {
        CORBA_string                    aDescription ;
                                        /* max length 255 chars */
        DSM_AppNPT                      duration ;
        CORBA_boolean                   audio ;
        CORBA_boolean                   video ;
        CORBA_boolean                   data ;
        }
        DSM_Stream_Info_T ;

#define DSM_Stream_Info_get_ACR         DSM_READER
#define DSM_Stream_Info_put_ACR         DSM_OWNER

DSM_Stream_Info_T     * DSM_Stream__get_Info
                                ( DSM_Stream            o,
                                CORBA_Environment     * ev ) ;
void                    DSM_Stream__set_Info
                                ( DSM_Stream            o,
                                DSM_Stream_Info_T     * Info,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the stream whose information is to be returned or set
Info	*(in)* the new value of the information
ev	*(in/out)* CORBA environment

Returns

DSM_Stream__get_Info returns DSM_Stream_Info_T
DSM_Stream__set_Info returns void

Exceptions

(standard)

Description

Returns or sets the information attribute of a stream.

Notes

1. DSM_Stream__get_Info requires READER privilege. DSM_Stream__set_Info requires OWNER privilege.

2. It is suggested that aDescription include the title of the Stream.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.1.

DSM::Stream::jump − when stream reaches stop NPT, resume at start NPT

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_jump_ACR             DSM_READER

DSM_RequestHandle       DSM_Stream_jump
                                ( DSM_Stream            o,
                                DSM_AppNPT            * rStart,
                                DSM_AppNPT            * rStop,
                                DSM_Scale             * rScale,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the stream being played
rStart	*(in)* the starting time
rStop	*(in)* the stopping time
rScale	*(in)* the rate and direction of play
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_SCALE
DSM::BAD_START
DSM::BAD_STOP
DSM::MPEG_DELIVERY

Description

When the stream reaches rStop, resume play at rStart with rate and direction rScale.

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.9.

DSM::Stream::Lock − status of stream read and write locks

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_Lock_T               DSM_Access_Lock_T

#define DSM_Stream_Lock_get_ACR         DSM_Access_Lock_get_ACR
#define DSM_Stream_Lock_put_ACR         DSM_Access_Lock_put_ACR

#define DSM_Stream__get_Lock            DSM_Access__get_Lock
#define DSM_Stream__set_Lock            DSM_Access__set_Lock
```

Description

Returns or sets the lock attribute of a stream. Inherited from, and the same as, `DSM::Access::Lock` (*q.v.*).

Notes

1. `DSM_Stream__get_Lock` requires `READER` privilege. `DSM_Stream__set_Lock` requires `WRITER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.

DSM::Stream::pause − stop sending stream when NPT is reached

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_pause_ACR            DSM_READER

DSM_RequestHandle       DSM_Stream_pause
                                ( DSM_Stream            o,
                                DSM_AppNPT            * rStop,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the stream being played
rStop	*(in)* the stopping time
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_STOP
DSM::MPEG_DELIVERY

Description

When the stream reaches rStop, halt play.

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.5.

DSM::Stream::Perms − access permission information for a stream

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_Perms_T              DSM_Access_Perms_T

#define DSM_Stream_Perms_get_ACR        DSM_Access_Perms_get_ACR
#define DSM_Stream_Perms_put_ACR        DSM_Access_Perms_put_ACR

#define DSM_Stream__get_Perms           DSM_Access__get_Perms
#define DSM_Stream__set_Perms           DSM_Access__set_Perms
```

Description

Returns or sets the permission attribute of a stream. Inherited from, and the same as, `DSM::Access::Perms` (*q.v.*).

Notes

1. Both of these operations require `OWNER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3

DSM::Stream::play − play stream from start NPT until stop NPT

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_play_ACR             DSM_READER

DSM_RequestHandle       DSM_Stream_play
                                ( DSM_Stream            o,
                                DSM_AppNPT            * rStart,
                                DSM_AppNPT            * rStop,
                                DSM_Scale             * rScale,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the stream being played
rStart	*(in)* the starting time
rStop	*(in)* the stopping time
rScale	*(in)* the rate and direction of play
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_SCALE
DSM::BAD_START
DSM::BAD_STOP
DSM::MPEG_DELIVERY

Description

Begin playing at rStart with rate and direction rScale, halting when rStop is reached.

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.10.

DSM::Stream::reset − reset a stream state machine

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_reset_ACR            DSM_READER

DSM_RequestHandle       DSM_Stream_reset
                                ( DSM_Stream            o,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the stream being played
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

(standard exceptions)

Description

Reset the stream state machine.

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.8.

DSM::Stream::resume − start sending stream at NPT

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_resume_ACR           DSM_READER

DSM_RequestHandle       DSM_Stream_resume
                                ( DSM_Stream            o,
                                DSM_AppNPT            * rStart,
                                DSM_Scale             * rScale,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the stream being played
rStart	*(in)* the starting time
rScale	*(in)* the rate and direction of play
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::BAD_SCALE
DSM::BAD_START
DSM::MPEG_DELIVERY

Description

Resume play at rStart with rate and direction rScale.

Notes

1. This operation requires READER privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.6.

DSM::Stream::Size − size of a stream

Synopsis − C

```
#include    <dsmcc.h>

#define DSM_Stream_Size_get_ACR         DSM_Access_Size_get_ACR

#define DSM_Stream__get_Size            DSM_Access__get_Size
```

Description

Returns the size of a stream's attributes. Inherited from, and the same as, `DSM::Access::Size` (*q.v.*).

Notes

1. This operation requires `READER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.

DSM::Stream::status − obtain status of a stream

Synopsis − C

```
#include    <dsmcc.h>

typedef DSM_u_long                      DSM_Stream_Mode ;

/* mode values */
#define DSM_Stream_OPEN_M                       0UL
#define DSM_Stream_PAUSE_M                      1UL
#define DSM_Stream_TRANSPORT_M                  2UL
#define DSM_Stream_TRANSPORT_PAUSE_M            3UL
#define DSM_Stream_SEARCH_TRANSPORT_M           4UL
#define DSM_Stream_SEARCH_TRANSPORT_PAUSE_M     5UL
#define DSM_Stream_PAUSE_SEARCH_TRANSPORT_M     6UL
#define DSM_Stream_END_OF_STREAM_M              7UL
#define DSM_Stream_PRE_SEARCH_TRANSPORT_M       8UL
#define DSM_Stream_PRE_SEARCH_TRANSPORT_PAUSE_M 9UL

typedef struct DSM_Stream_Stat
        {
        DSM_AppNPT                      rPosition ;
        DSM_Scale                       rScale ;
        DSM_Stream_Mode                 aMode ;
        }
        DSM_Stream_Stat ;

#define DSM_Stream_status_ACR           DSM_READER

DSM_RequestHandle       DSM_Stream_status
                                ( DSM_Stream            o,
                                DSM_Stream_Stat       * rAppStatus,
                                DSM_Stream_Stat      ** rActStatus,
                                CORBA_Environment     * ev ) ;
```

Arguments

o	*(in)* the stream being played
rAppStatus	*(in)* estimated status
rActStatus	*(out)* actual status
ev	*(in/out)* CORBA environment

Returns

DSM_RequestHandle

Exceptions

DSM::MPEG_DELIVERY

Description

Given an (optional) estimate of current stream status, this operation returns the actual value.

Notes

1. This operation requires `READER` privilege.

Availability

spain

Reference

ISO/IEC 13818−6:1998(E) *Digital Storage Media Command & Control*, Sept 1st 1998, Section 5.5.1.3.7.

DSM::View − relational view of data

documentation to be supplied in a future release

Index

CosNaming 20
CosNaming_Binding 20
CosNaming_BindingIterator 23
CosNaming_BindingIterator_destroy 24
CosNaming_BindingIterator_next_n 25
CosNaming_BindingIterator_next_one 27
CosNaming_BindingList 21
CosNaming_BindingType 20
CosNaming_Istring 20
CosNaming_Name 20
CosNaming_NameComponent 20
CosNaming_NamingContext 28
CosNaming_NamingContext_AlreadyBound 28
CosNaming_NamingContext_bind 31
CosNaming_NamingContext_bind_context 33
CosNaming_NamingContext_bind_new_context 35
CosNaming_NamingContext_CannotProceed 28
CosNaming_NamingContext_destroy 37
CosNaming_NamingContext_InvalidName 28
CosNaming_NamingContext_list 38
CosNaming_NamingContext_missing_node 28
CosNaming_NamingContext_new_context 40
CosNaming_NamingContext_not_context 28
CosNaming_NamingContext_not_object 28
CosNaming_NamingContext_NotEmpty 29
CosNaming_NamingContext_NotFound 28
CosNaming_NamingContext_NotFoundReason 28
CosNaming_NamingContext_rebind 41
CosNaming_NamingContext_rebind_context 43
CosNaming_NamingContext_resolve 45
CosNaming_NamingContext_unbind 46
CosNaming_ncontext 20
CosNaming_nobject 20
CosNaming::BindingIterator 23
CosNaming::BindingIterator::destroy 24
CosNaming::BindingIterator::next_n 25
CosNaming::BindingIterator::next_one 27
CosNaming::NamingContext 28
CosNaming::NamingContext::AlreadyBound 28
CosNaming::NamingContext::bind 31
CosNaming::NamingContext::bind_context 33
CosNaming::NamingContext::bind_new_context 35
CosNaming::NamingContext::CannotProceed 28
CosNaming::NamingContext::destroy 37
CosNaming::NamingContext::InvalidName 28
CosNaming::NamingContext::list 38
CosNaming::NamingContext::new_context 40
CosNaming::NamingContext::NotEmpty 29
CosNaming::NamingContext::NotFound 28
CosNaming::NamingContext::rebind 41

CosNaming::NamingContext::rebind_context 43
CosNaming::NamingContext::resolve 45
CosNaming::NamingContext::unbind 46
DSM_Access 47
DSM_Access__get_Hist 48
DSM_Access__get_Lock 51
DSM_Access__get_Perms 53
DSM_Access__get_Size 56
DSM_Access__set_Hist 48
DSM_Access__set_Lock 51
DSM_Access__set_Perms 53
DSM_Access_Hist_T 48
DSM_Access_Lock_T 51
DSM_Access_Perms_T 53
DSM_AuthRequest 138
DSM_AuthRequest_T 138
DSM_Base 57
DSM_Base_close 58
DSM_Base_destroy 60
DSM_CompatibilityDescriptor 61
DSM_Config__get_DeferredSync 65
DSM_Config__set_DeferredSync 65
DSM_Config_inquire 67
DSM_Config_wait 68
DSM_DateTime 48
DSM_Directory 69
DSM_Directory__get_Hist 79
DSM_Directory__get_Lock 81
DSM_Directory__get_Perms 85
DSM_Directory__get_Size 91
DSM_Directory__set_Hist 79
DSM_Directory__set_Lock 81
DSM_Directory__set_Perms 85
DSM_Directory_bind 71
DSM_Directory_bind_context 72
DSM_Directory_bind_new_context 73
DSM_Directory_close 74
DSM_Directory_destroy 76
DSM_Directory_get 77
DSM_Directory_list 80
DSM_Directory_new_context 82
DSM_Directory_open 83
DSM_Directory_put 86
DSM_Directory_rebind 88
DSM_Directory_rebind_context 89
DSM_Directory_resolve 90
DSM_Directory_unbind 92
DSM_Download 93
DSM_Download_alloc 95
DSM_Download_cancel 97
DSM_Download_info 98
DSM_Download_start 100
DSM_DownloadSI 101
DSM_DownloadSI_cancel 102
DSM_DownloadSI_CancelRequest 102
DSM_DownloadSI_DataRequest 110

DSM_DownloadSI_deinstall 105
DSM_DownloadSI_info 106
DSM_DownloadSI_install 108
DSM_DownloadSI_ModuleInstallInfo 108
DSM_DownloadSI_ModuleInstallList 108
DSM_DownloadSI_proceed 110
DSM_Event 112
DSM_Event__get_EventList 113
DSM_Event__put_EventList 113
DSM_Event_notify 115
DSM_Event_subscribe 116
DSM_Event_unsubscribe 117
DSM_EventList_T 113
DSM_eventName 113
DSM_File 118
DSM_File__get_Content 120
DSM_File__get_ContentSize 121
DSM_File__get_Hist 123
DSM_File__get_Lock 124
DSM_File__get_Perms 125
DSM_File__get_Size 128
DSM_File__put_Content 120
DSM_File__set_Hist 123
DSM_File__set_Lock 124
DSM_File__set_Perms 125
DSM_File_close 119
DSM_File_destroy 122
DSM_File_read 126
DSM_File_write 129
DSM_IFKind 133
DSM_IFKindList 133
DSM_ik_* 133
DSM_InfoRequest 106
DSM_InfoResponse 106
DSM_InterfaceDescriptor 61
DSM_IntfCode 133
DSM_Kind 133
DSM_Kind_has_a 135
DSM_Kind_is_a 136
DSM_ModuleInfo 93
DSM_ModuleInfoList 93
DSM_ModuleList 93
DSM_ObjRefs 147
DSM_PathSpec 69
DSM_PathType 69
DSM_PathValues 69
DSM_RequestHandle 65
DSM_Security 138
DSM_Security_authenticate 139
DSM_ServiceDomain 147
DSM_ServiceGateway 140
DSM_ServiceGateway__get_Hist 140
DSM_ServiceGateway__get_Lock 140
DSM_ServiceGateway__get_Perms 140
DSM_ServiceGateway__get_Size 140
DSM_ServiceGateway__set_Hist 141

DSM_ServiceGateway__set_Lock 141
DSM_ServiceGateway__set_Perms 141
DSM_ServiceGateway_attach 140
DSM_ServiceGateway_bind 140
DSM_ServiceGateway_bind_context 140
DSM_ServiceGateway_bind_new_context 140
DSM_ServiceGateway_close 140
DSM_ServiceGateway_destroy 140
DSM_ServiceGateway_detach 140
DSM_ServiceGateway_get 140
DSM_ServiceGateway_list 141
DSM_ServiceGateway_new_context 141
DSM_ServiceGateway_put 141
DSM_ServiceGateway_rebind 141
DSM_ServiceGateway_rebind_context 141
DSM_ServiceGateway_resolve 141
DSM_ServiceGateway_unbind 141
DSM_ServiceGatewaySI 142
DSM_ServiceGatewaySI__get_Hist 142
DSM_ServiceGatewaySI__get_Lock 142
DSM_ServiceGatewaySI__get_Perms 142
DSM_ServiceGatewaySI__get_Size 142
DSM_ServiceGatewaySI__set_Hist 143
DSM_ServiceGatewaySI__set_Lock 143
DSM_ServiceGatewaySI__set_Perms 143
DSM_ServiceGatewaySI_attach 142
DSM_ServiceGatewaySI_bind 142
DSM_ServiceGatewaySI_bind_context 142
DSM_ServiceGatewaySI_bind_new_context 142
DSM_ServiceGatewaySI_close 142
DSM_ServiceGatewaySI_destroy 142
DSM_ServiceGatewaySI_detach 142
DSM_ServiceGatewaySI_get 142
DSM_ServiceGatewaySI_list 143
DSM_ServiceGatewaySI_new_context 143
DSM_ServiceGatewaySI_put 143
DSM_ServiceGatewaySI_rebind 143
DSM_ServiceGatewaySI_rebind_context 143
DSM_ServiceGatewaySI_resolve 143
DSM_ServiceGatewaySI_unbind 143
DSM_ServiceGatewayUU 144
DSM_ServiceGatewayUU__get_Hist 144
DSM_ServiceGatewayUU__get_Lock 144
DSM_ServiceGatewayUU__get_Perms 144
DSM_ServiceGatewayUU__get_Size 144
DSM_ServiceGatewayUU__set_Hist 145
DSM_ServiceGatewayUU__set_Lock 145
DSM_ServiceGatewayUU__set_Perms 145
DSM_ServiceGatewayUU_bind 144
DSM_ServiceGatewayUU_bind_context 144
DSM_ServiceGatewayUU_bind_new_context 144
DSM_ServiceGatewayUU_close 144
DSM_ServiceGatewayUU_destroy 144
DSM_ServiceGatewayUU_get 144
DSM_ServiceGatewayUU_list 144
DSM_ServiceGatewayUU_new_context 144

DSM_ServiceGatewayUU_put 145
DSM_ServiceGatewayUU_rebind 145
DSM_ServiceGatewayUU_rebind_context 145
DSM_ServiceGatewayUU_resolve 145
DSM_ServiceGatewayUU_unbind 145
DSM_Session 146
DSM_Session_attach 147
DSM_Session_detach 149
DSM_SessionSI 150
DSM_SessionSI_attach 151
DSM_SessionSI_detach 153
DSM_SessionUU_attach 155
DSM_SessionUU_detach 157
DSM_State 159
DSM_State_resume 160
DSM_Step 69
DSM_Stream__get_Hist 169
DSM_Stream__get_Info 170
DSM_Stream__get_Lock 173
DSM_Stream__get_Perms 175
DSM_Stream__get_Size 179
DSM_Stream__set_Hist 169
DSM_Stream__set_Info 170
DSM_Stream__set_Lock 173
DSM_Stream__set_Perms 175
DSM_Stream_*_M 180
DSM_Stream_close 167
DSM_Stream_destroy 168
DSM_Stream_Info_T 170
DSM_Stream_jump 172
DSM_Stream_Lock_T 173
DSM_Stream_Mode 180
DSM_Stream_pause 174
DSM_Stream_Perms_T 175
DSM_Stream_play 176
DSM_Stream_reset 177
DSM_Stream_resume 178
DSM_Stream_Stat 180
DSM_Stream_status 180
DSM_SubDescriptor 61
DSM_UserContext 147
DSM_Version 48
DSM::Access 47
DSM::Access::Hist 48
DSM::Access::Lock 51
DSM::Access::Perms 53
DSM::Access::Size 56
DSM::Base 57
DSM::Base::close 58
DSM::Base::destroy 60
DSM::CompatibilityDescriptor 61
DSM::Composite 64
DSM::Config 65
DSM::Config::DeferredSync 65
DSM::Config::inquire 67
DSM::Config::wait 68

DSM::Directory 69
DSM::Directory::bind 71
DSM::Directory::bind_context 72
DSM::Directory::bind_new_context 73
DSM::Directory::close 74
DSM::Directory::destroy 76
DSM::Directory::get 77
DSM::Directory::Hist 79
DSM::Directory::list 80
DSM::Directory::Lock 81
DSM::Directory::new_context 82
DSM::Directory::open 83
DSM::Directory::Perms 85
DSM::Directory::put 86
DSM::Directory::rebind 88
DSM::Directory::rebind_context 89
DSM::Directory::resolve 90
DSM::Directory::Size 91
DSM::Directory::unbind 92
DSM::Download 93
DSM::Download::alloc 95
DSM::Download::cancel 97
DSM::Download::info 98
DSM::Download::start 100
DSM::DownloadSI 101
DSM::DownloadSI::cancel 102
DSM::DownloadSI::deinstall 105
DSM::DownloadSI::info 106
DSM::DownloadSI::install 108
DSM::DownloadSI::proceed 110
DSM::Event 112
DSM::Event::EventList 113
DSM::Event::notify 115
DSM::Event::subscribe 116
DSM::Event::unsubscribe 117
DSM::File 118
DSM::File::close 119
DSM::File::Content 120
DSM::File::ContentSize 121
DSM::File::destroy 122
DSM::File::Hist 123
DSM::File::Lock 124
DSM::File::Perms 125
DSM::File::read 126
DSM::File::Size 128
DSM::File::write 129
DSM::First 131
DSM::Interfaces 132
DSM::Kind 133
DSM::Kind::has_a 135
DSM::Kind::is_a 136
DSM::LifeCycle 137
DSM::Security 138
DSM::Security::authenticate 139
DSM::ServiceGateway 140
DSM::ServiceGateway::attach 140

DSM::ServiceGateway::bind 140
DSM::ServiceGateway::bind_context 140
DSM::ServiceGateway::bind_new_context 140
DSM::ServiceGateway::close 140
DSM::ServiceGateway::destroy 140
DSM::ServiceGateway::detach 140
DSM::ServiceGateway::get 140
DSM::ServiceGateway::Hist 141
DSM::ServiceGateway::list 141
DSM::ServiceGateway::Lock 141
DSM::ServiceGateway::new_context 141
DSM::ServiceGateway::Perms 141
DSM::ServiceGateway::put 141
DSM::ServiceGateway::rebind 141
DSM::ServiceGateway::rebind_context 141
DSM::ServiceGateway::resolve 141
DSM::ServiceGateway::Size 141
DSM::ServiceGateway::unbind 141
DSM::ServiceGatewaySI 142
DSM::ServiceGatewaySI::attach 142
DSM::ServiceGatewaySI::bind 142
DSM::ServiceGatewaySI::bind_context 142
DSM::ServiceGatewaySI::bind_new_context 142
DSM::ServiceGatewaySI::close 142
DSM::ServiceGatewaySI::destroy 142
DSM::ServiceGatewaySI::detach 142
DSM::ServiceGatewaySI::get 142
DSM::ServiceGatewaySI::Hist 143
DSM::ServiceGatewaySI::list 143
DSM::ServiceGatewaySI::Lock 143
DSM::ServiceGatewaySI::new_context 143
DSM::ServiceGatewaySI::Perms 143
DSM::ServiceGatewaySI::put 143
DSM::ServiceGatewaySI::rebind 143
DSM::ServiceGatewaySI::rebind_context 143
DSM::ServiceGatewaySI::resolve 143
DSM::ServiceGatewaySI::Size 143
DSM::ServiceGatewaySI::unbind 143
DSM::ServiceGatewayUU 144
DSM::ServiceGatewayUU::bind 144
DSM::ServiceGatewayUU::bind_context 144
DSM::ServiceGatewayUU::bind_new_context 144
DSM::ServiceGatewayUU::close 144
DSM::ServiceGatewayUU::destroy 144
DSM::ServiceGatewayUU::get 144
DSM::ServiceGatewayUU::Hist 144
DSM::ServiceGatewayUU::list 144
DSM::ServiceGatewayUU::Lock 144
DSM::ServiceGatewayUU::new_context 144
DSM::ServiceGatewayUU::Perms 145
DSM::ServiceGatewayUU::put 145
DSM::ServiceGatewayUU::rebind 145
DSM::ServiceGatewayUU::rebind_context 145
DSM::ServiceGatewayUU::resolve 145
DSM::ServiceGatewayUU::Size 145
DSM::ServiceGatewayUU::unbind 145

DSM::Session 146
DSM::Session::attach 147
DSM::Session::detach 149
DSM::SessionSI 150
DSM::SessionSI::attach 151
DSM::SessionSI::detach 153
DSM::SessionUU 154
DSM::SessionUU::attach 155
DSM::SessionUU::detach 157
DSM::State 159
DSM::State::resume 160
DSM::State::suspend 161
DSM::Stream 162
DSM::Stream::close 167
DSM::Stream::destroy 168
DSM::Stream::Hist 169
DSM::Stream::Info 170
DSM::Stream::jump 172
DSM::Stream::Lock 173
DSM::Stream::pause 174
DSM::Stream::Perms 175
DSM::Stream::play 176
DSM::Stream::reset 177
DSM::Stream::resume 178
DSM::Stream::Size 179
DSM::Stream::status 180
DSM::View 182