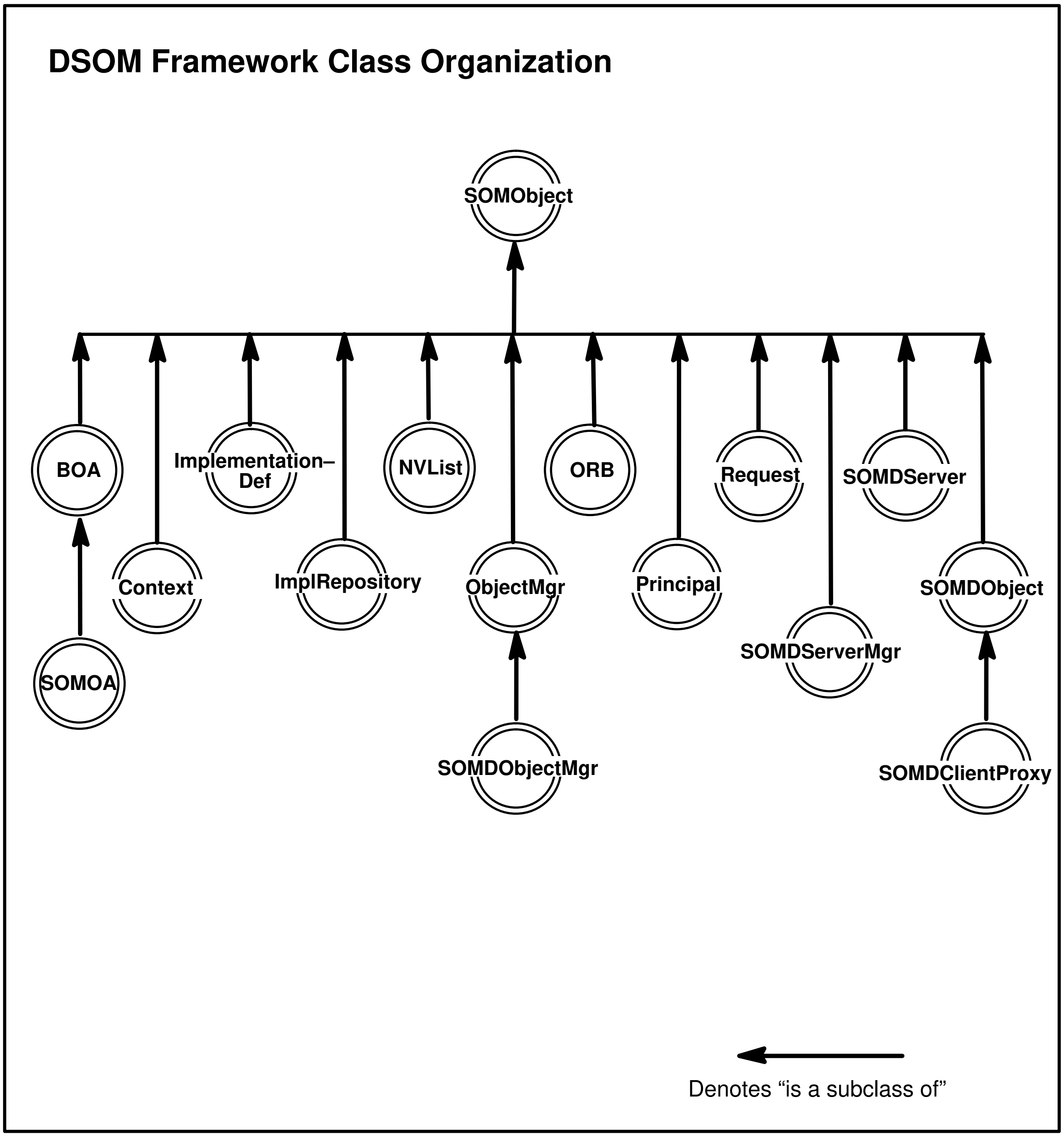
DSOM Framework Reference



Notes

DSOM and CORBA

Distributed SOM (DSOM) is a framework which supports access to objects in a distributed application. DSOM can be viewed as both:

- an extension to basic SOM facilities
- an implementation of the "Object Request Broker" (ORB) technology defined by the Object Management Group (OMG), in the Common Object Request Broker Architecture (CORBA) specification and standard, Revision 1.1. The CORBA 1.1 specification is published by x/Open and the Object Management Group (OMG).

One of the primary contributions of CORBA is the specification of basic runtime interfaces for writing portable, distributable object-oriented applications. SOM and DSOM implement those runtime interfaces, according to the CORBA specification.

In addition to the published CORBA 1.1 interfaces, it was necessary for DSOM to introduce several of its own interfaces, in those areas where:

- CORBA 1.1 did not specify the full interface (for example, **ImplementationDef**, **Principal**),
- CORBA 1.1 did not address the function specified by the interface (for example, "lifecycle" services for object creation and deletion), or
- the functionality of a CORBA 1.1 interface has been enhanced by DSOM.

Any such interfaces have been noted on the reference page for each DSOM *class*.

A note on method naming conventions

The SOM Toolkit frameworks (including DSOM) and CORBA have slightly different conventions for naming methods. Methods introduced by the SOM Toolkit frameworks use prefixes to indicate the framework to which each method belongs, and use capitalization to separate words in the method names (for example, **somdFindServer**). Methods introduced by CORBA have no prefixes, are all lower case, and use underscores to separate words in the method names (such as, **impl_is_ready**).

DSOM, more than the other SOM Toolkit frameworks, uses a mix of both conventions. The method and class names introduced by CORBA 1.1 are implemented as specified, for application portability. Methods introduced by DSOM to enhance a CORBA-defined class also use the CORBA naming style. The SOM Toolkit convention for method naming is used for non-CORBA classes which are introduced by DSOM.

get_next_response Function

Purpose

Returns the next **Request** object to complete, after starting multiple requests in parallel.

C Syntax

ORBStatus get_next_response (
Environment* *env,*
Flags *response_flags,*
Request ******req* **);**

Description

The **get_next_response** function returns a pointer to the next **Request** object to complete after starting multiple requests in parallel. This is a synchronization function used in conjunction with the **send_multiple_requests** function. There is no specific order in which requests will complete.

If the *response_flags* field is set to 0, this function will not return until the next request completion. If the caller does not want to become blocked, the RESP_NO_WAIT flag should be specified.

Parameters

env	A pointer to the **Environment** structure for the caller.
response_flags	A **Flags** (unsigned long) variable, used to indicate whether the caller wants to wait for the next request to complete (0), or not wait (RESP_NO_WAIT).
req	A pointer to a **Request** object variable. The address of the next **Request** object which completes is returned in the **Request** variable.

Return Value

The **get_next_response** function may return a non-zero **ORBStatus** value, which indicates a DSOM error code. (DSOM error codes are listed in Appendix A of the *SOM Toolkit User's Guide.*)

Example

See the example for the **send_multiple_requests** function.

Related Information

Functions: send_multiple_requests
Methods: send, get_response, invoke

This function is described in section 6.3, "Deferred Synchronous Routines", of the CORBA 1.1 specification.

ORBfree Function

Purpose

Frees memory allocated by DSOM for return values and **out** arguments.

C Syntax

void ORBfree (void* *ptr***);**

Description

The **ORBfree** function is used to free memory for method return values or **out** arguments which are placed in memory allocated by DSOM (versus the calling program). For example, strings, arrays, sequence buffers, and "any" values are returned in memory which is dynamically allocated by DSOM.

Parameters

ptr	A pointer to memory that has been dynamically allocated by DSOM for a method return value or **out** argument.

Return Value

None.

Example

```
#include <somd.h>
#include <myobject.h>   /* provided by user */

MyObject obj;
Environment ev;
string str;

/* assume myMethod has the following IDL declaration
 * in the MyObject interface:
 *
 *    void myMethod(out string s);
 */
_myMethod(obj, &ev, &str);
...

/* free storage */
ORBfree(str);
```

Related Information

Functions: SOMD_NoORBfree

This function is described in section 5.16, "Argument Passing Considerations", and section 5.17, "Return Result Passing Considerations", of the CORBA 1.1 specification.

send_multiple_requests Function

Purpose

Initiates multiple **Requests** in parallel.

C Syntax

ORBStatus send_multiple_requests (
Request *reqs[ ],*
Environment* *env,*
long *count,*
Flags *invoke_flags* **);**

Description

The **send_multiple_requests** function initiates multiple **Requests** "in parallel". (The actual degree of parallelism is system dependent.) Each **Request** object is created using the **create_request** method, defined on **SOMDClientProxy**. Like the **send** method, this function returns to the caller immediately without waiting for the **Requests** to finish. The caller waits for the request responses using the **get_next_response** function.

Parameters

reqs — The address of an array of **Requests** objects which are to be initiated in parallel.

env — A pointer to the **Environment** structure for the caller.

count — The number of **Request** objects in *reqs.*

invoke_flags — A **Flags** (unsigned long) value, used to indicate the following options:

INV_NO_RESPONSE –	Indicates the caller does not intend to get any results or **out** parameter values from any of the requests. The requests can be treated as if they are **oneway** operations.
INV_TERM_ON_ERR –	If one of the requests causes an error, the remaining requests are not sent.

The above flag values may be "or"-ed together.

Return Value

The **send_multiple_requests** function may return a non-zero **ORBStatus** value, which indicates a DSOM error code. (DSOM error codes are listed in Appendix A of the *SOM Toolkit User's Guide.*)

Example

```
#include <somd.h>

/* sum a set of values in parallel */
int parallel_sum(Environment *ev, int n, SOMDObject *objs)
{
   int index, sum = 0;
   Request *next;
   Request *reqs = (Request*) SOMMalloc(n * sizeof(Request));
   NamedValue *results = (NamedValue*)
                       SOMMalloc(n * sizeof(Namedvalue));

   for (i=0; i < n; i++)
      (void) _create_request((Context *)NULL, "_get_count", NULL,
                         &(result[i]), &(reqs[i]), (Flags)0);

   (void) send_multiple_requests(reqs, ev, n, (Flags)0);

   for (i=0, i < n; i++) {
      (void) get_next_response(ev, (Flags)0, &next);
      index = (next - reqs);
      sum += *((int*)results[index].argument._value);
   }

   return(sum);
}
```

Related Information

Functions: get_next_response
Methods: send, get_response, invoke

This function is described in section 6.3, “Deferred Synchronous Routines”, of the CORBA 1.1 specification.

somdExceptionFree Function

Purpose

Frees the memory held by the exception structure within an **Environment** structure, regardless of whether the exception was returned by a local or a remote method call.

C Syntax

void somdExceptionFree (Environment *_ev_);

Description

The **somdExceptionFree** function frees the memory held by the exception structure within an **Environment** structure, regardless of whether the exception was returned by a local or a remote method call.

When a DSOM client program invokes a remote method and the method returns an exception in the **Environment** structure, it is the client's responsibility to free the exception. This is done by calling either **exception_free** or **somdExceptionFree** on the **Environment** structure in which the exception was returned. (The two functions are equivalent. The **exception_free** function name is #defined in the som.h or som.xh file to provide strict CORBA compliance of function names.) There is a similar function, **somExceptionFree**, available for SOM programmers; DSOM programmers, however, can use **somdExceptionFree** to free all exceptions (regardless of whether they were returned from a local or a remote method call).

Parameters

_ev_	The **Environment** structure whose exception information is to be freed.

Return Value

None.

Example

```
X_foo(x, ev, 23);  /* make a remote method call */
if (ev->major != NO_EXCEPTION)
{
    printf("foo exception = %s\n", somExceptionId(ev));

    /* ... handle exception ... */

    somdExceptionFree(ev);  /* free exception */
}
```

Related Information

Functions: somExceptionFree, somExceptionId, somExceptionValue, somSetException (all SOM kernel functions)

Data structures: Environment (somcorba.h**)**

SOMD_Init Function

Purpose

Initializes DSOM in the calling process.

C Syntax

void SOMD_Init (Environment* *env***);**

Description

Initializes DSOM in the calling process. This function should be called before any other DSOM functions or methods. This function should only be invoked (a) at the beginning of a DSOM program (client or server), to initialize the program, or (b) after **SOMD_Uninit** has been invoked, to reinitialize the program. If the program has already been initialized with **SOMD_Init**, then invoking **SOMD_Init** again has no effect.

An effect of calling **SOMD_Init** is that the global variables **SOMD_ObjectMgr, SOMD_ImplRepObject,** and **SOMD_ORBObject**, are initialized with pointers to the (single) instances of the **SOMDObjectMgr, ImplRepository,** and **ORB** objects.

Parameters

env — A pointer to the **Environment** structure for the caller.

Return Value

None. (However, the global variables **SOMD_ObjectMgr, SOMD_ImplRepObject,** and **SOMD_ORBObject** are set implicitly.)

Example

```
#include <somd.h>

Environment ev;

/* initialize Environment */
SOM_InitEnvironment(&ev);

/* initialize DSOM runtime */
SOMD_Init(&ev);
...

/* Free DSOM resources */
SOMD_Uninit(&ev);
```

Related Information

See Chapter 6 on DSOM in the *SOM Toolkit User's Guide.*

SOMD_NoORBfree Function

Purpose

Specifies to DSOM that the client program will use the **SOMFree** function to free memory allocated by DSOM, rather than using the **ORBfree** function.

C Syntax

void SOMD_NoORBfree ();

Description

The **SOMD_NoORBfree** function is used in a DSOM client program to specify to DSOM that the client program will use the **SOMFree** function to free memory allocated by DSOM, rather than using the **ORBfree** function.

Typically, a DSOM client program will use **SOMFree** to free memory returned from local method calls and **ORBfree** to free memory returned from remote method calls. The **SOMD_NoORBfree** function allows programmers to use a single function (**SOMFree**) to free blocks of memory, regardless of whether they were allocated locally or by DSOM in response to a remote method call.

SOMD_NoORBfree, if used, should be called just after calling **SOMD_Init** in the client program. In response to this call, DSOM will not keep track of the memory it allocates for the client. Instead, it will assume that the client program will be responsible for walking all data structures returned from remote method calls, calling **SOMFree** for each block of memory within.

Parameters

None.

Return Value

None.

Example

```
SOMD_Init();
SOMD_NoORBfree();

/* rest of client program */
```

Related Information

Functions: ORBfree, SOMFree

SOMD_RegisterCallback Function

Purpose

Registers a callback function for handling DSOM request events.

C Syntax

void SOMLINK SOMD_RegisterCallback (SOMEEMan *emanObj*, **EMRegProc ***func***);**

Description

When writing event-driven applications where there are event sources other than DSOM requests (for example, user input, mouse clicks, and so forth), DSOM cannot be given exclusive control of the "main loop," such as when **execute_request_loop** is called. Instead, the application should use the Event Management (EMan) framework to register and process all application events.

The **SOMD_RegisterCallback** function is used to register a user-supplied DSOM event handler function with EMan. The caller need only supply an address of the event handler function, and the instance of the EMan object — the details of registration are implemented by **SOMD_RegisterCallback**.

Callback functions should have the SOMLINK keyword explicitly specified, except on Windows. Using an explicit SOMLINK keyword on Windows will preclude the ability of an application to support multiple instances.

Note: The function **SOMD_RegisterCallback** must be declared with "system linkage" on OS/2.

Parameters

emanObj A pointer to an instance of **SOMEEman**, the Event Manager object.

func A pointer to an event handler function which will be called by EMan whenever a DSOM request arrives. This function must have the following prototype (equivalent to the **EMRegProc** type defined in "eman.h"):

```
#ifdef __OS2__
 #pragma linkage(func, system)
#endif

void SOMLINK func (SOMEEvent event, void *eventData)
/* On Windows, using the SOMLINK keyword precludes
 * the support of multiple instances. */
```

Return Value

None.

Example

```
#include <somd.h>
#include <eman.h>

#ifdef __OS2__
 #pragma linkage(SOMD_RegisterCallback, system)
 #pragma linkage(DSOMEventCallBack, system)
#endif

/* On Windows, this example would omit the SOMLINK keyword. */

void SOMLINK DSOMEventCallBack (SOMEEvent event, void *eventData)
{
   Environment ev;
   SOM_InitEnvironment(&ev);

   _execute_request_loop (SOMD_SOMOAObject, &ev, SOMD_NO_WAIT);
}

main()
{
   ...
   eman = SOMEEmanNew();
   SOMD_RegisterCallback(eman, DSOMEventCallBack);

   _someProcessEvents(eman, &ev);   /* main loop */
   ...
}
```

Related Information

See Chapter 12 of the *SOM Toolkit User's Guide* for a description of the Event Management (EMan) framework, for writing event-driven applications.

SOMD_Uninit Function

Purpose

Free system resources allocated for use by DSOM.

C Syntax

void SOMD_Uninit (Environment* *env***);**

Description

Frees system resources (such as, shared memory segments, semaphores) allocated to the calling process for use by DSOM. This function should be called before a process exits, to ensure system resources are reused.

No DSOM functions or methods should be called after **SOMD_Uninit** has been called. After **SOMD_Uninit** is called, the program can be reinitialized by calling **SOMD_Init**. (**SOMD_Uninit** would then need to be called again before program termination, to uninitialize the program.)

Parameters

env	A pointer to the **Environment** structure for the caller.

Return Value

None.

Example

```
#include <somd.h>

Environment ev;

/* initialize Environment */
SOM_InitEnvironment(&ev);

/* initialize DSOM runtime */
SOMD_Init(&ev);
...
/* Free DSOM resources */
SOMD_Uninit(&ev);
```

Related Information

See Chapter 6 on DSOM in the *SOM Toolkit User's Guide.*

Context_delete Macro

Purpose

Deletes a **Context** object.

Syntax

ORBStatus Context_delete (
Context *ctxobj*,
Environment **env*,
Flags *del_flag***);**

Description

The **Context_delete** macro deletes the specified **Context** object. This macro maps to the **destroy** method of the **Context** class.

Parameters

ctxobj — A pointer to the **Context** object to be deleted.

env — A pointer to the **Environment** structure for the caller.

del_flag — A bitmask (unsigned long). If the flag CTX_DELETE_DESCENDANTS is specified, the macro deletes the specified **Context** object and all of its descendant **Context** objects. A zero value indicates that the flag is not set.

Expansion

Context_destroy (*ctxobj, env, del_flag* **)**

Example

```
#include <somd.h>

Environment ev;
Context cxt, newcxt;
long rc;
...
/* get the process' default Context */
rc = _get_default_context(SOMD_ORBObject, &ev, &cxt);

/* make newcxt a child Context of the default Context (cxt) */
rc = _create_child(cxt, &ev, "myContext", &newcxt);
...
/* assuming no descendent Contexts have been
 * created from newcxt, we can destroy newcxt with flags=0
 */
rc = Context_delete(newcxt, &ev, (Flags) 0);
```

Related Information

Methods: Context_destroy

Request_delete Macro

Purpose

Deletes the memory allocated by the ORB for a **Request** object.

Syntax

ORBStatus Request_delete (
Request *reqobj*,
Environment ****env*);**

Description

The **Request_delete** macro deletes the specified **Request** object and all associated memory. This macro maps to the **destroy** method of the **Request** class.

Parameters

reqobj	A pointer to the **Request** object to be deleted.
env	A pointer to the **Environment** structure for the caller.

Expansion

Request_destroy (*reqobj, env* **)**

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code sends a request to execute the call:
 *      result = methodLong(fooObj, &ev, 100,200);
 * using the DII without waiting for the result. Then, later,
 * waits for and then uses the result.
 */
Environment ev;
NVList arglist;
long rc;
Foo fooObj;
Request reqObj;
NamedValue result;

/* see the Example code for invoke to see how the request
 * is built
 */

/* Create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, "methodLong",
                     arglist, &result, &reqObj, (Flags)0);

/* Finally, send the request */
rc = _send(reqObj, &ev, (Flags)0);

/* do some work, i.e. don't wait for the result */

/* wait here for the result of the request */
rc = _get_response(reqObj, &ev, (Flags)0);

/* use the result */
if (result->argument._value == 9600) {...}

/* throw away the reqObj */
Request_delete(reqObj, &ev);
```

Related Information

Methods: Request_destroy

BOA Class

Description

The Basic Object Adapter (**BOA**) defines the basic interfaces that a server process uses to access services of an Object Request Broker like DSOM. The **BOA** defines methods for creating and exporting object references, registering implementations, activating implementations and authenticating requests.

For more information on the Basic Object Adapter, refer to Chapter 9 in the CORBA 1.1 specification.

NOTE: DSOM treats the **BOA** interface as an *abstract* class, which merely defines basic runtime interfaces (introduced in the CORBA specification) but does not implement those interfaces. Thus, there is no point in instantiating a **BOA** object. If a **BOA** object is created, any methods invoked on it will return a NO_IMPLEMENT exception. Instead, the SOM Object Adapter (**SOMOA**) subclass provides DSOM implementations for **BOA** methods. When a **BOA** method is invoked on the **SOMOA** object, the desired behavior will occur.

File Stem

boa

Base

SOMObject

Metaclass

SOMMSingleInstance

Ancestor Classes

SOMObject

Subclasses

SOMOA

New Methods

change_implementation
create
deactivate_impl
deactivate_obj
dispose
get_id
get_principal
impl_is_ready
obj_is_ready
set_exception

change_implementation Method

Purpose

Changes the implementation associated with the referenced object. *(Not implemented.)*

IDL Syntax

void change_implementation (
in SOMDObject *obj***,**
in ImplementationDef *impl***);**

Description

The **change_implementation** method is defined by the CORBA specification, *but has a null implementation in DSOM.* This method always returns a NO_IMPLEMENT exception.

In CORBA 1.1, the **change_implementation** method is provided to allow an application to change the implementation definition of an object.

However, in DSOM, the **ImplementationDef** identifies the server which implements an object. In these terms, changing an object's implementation (that is, server) would result in a change in the object's location. In DSOM, moving objects from one server to another is considered an application-specific task, and hence, no default implementation is provided.

It *is* possible, however, to change the program which implements an object's server, or change the class library which implements an object's class. To modify the program associated with an **ImplementationDef**, use the **update_impldef** method defined on **ImplRepository.** To change the implementation of an object's class, replace the corresponding class library with a new (upward-compatible) one.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to the **SOMDObject** object which refers to the application object whose implementation is to be changed.
impl	A pointer to the **ImplementationDef** object representing the new implementation of the application object.

Return Value

The **SOMOA** implementation always returns a NO_IMPLEMENT exception, with a minor code of SOMDERROR_NotImplemented.

Original Class

BOA

create Method

Purpose

Creates a “reference” for a local application object which can be exported to remote clients.

IDL Syntax

```
typedef sequence<octet,1024> ReferenceData;        // in somdtype.idl

SOMDObject  create (
                        in ReferenceData id,
                        in InterfaceDef intf,
                        in ImplementationDef impl);
```

Description

The **create** method creates a **SOMDObject** which is used as a “reference” to a local application object. An object reference is simply an object which is used to refer to another target object — one may think of it as an “ID”, “link”, or “handle.” Object references are important in DSOM in that their values can be externalized (that is, can be represented in a string form) for transmission between processes, storage in files, and so on. In DSOM, the proxy objects in client processes are remote object references.

To create an object reference, the caller specifies the **ImplementationDef** of the calling process, the **InterfaceDef** of the target application object, and up to 1024 bytes of **ReferenceData** which is used by the application to identify and activate the application object. When subsequent method calls specify the object reference as a parameter, the application will use the reference to find and/or activate the referenced object.

Note that (as specified in CORBA 1.1) each call to **create** returns a unique object reference, even if the same parameters are used in subsequent calls. For each reference, the **ReferenceData** is stored in the reference data file (and backup file, if any) for the server.

The **SOMOA** class introduces a **change_id** method which allows a server to modify the **ReferenceData** of one of its references. (The **change_id** method is *not* in the CORBA 1.1 specification.)

Ownership of the returned **SOMDObject** is transferred to the caller.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
id	A pointer to the **ReferenceData** structure containing application-specific information describing the target object.
intf	A pointer to the **InterfaceDef** object which describes the interface of the target object.
impl	A pointer to the **ImplementationDef** object which describes the application (server) process which implements the target object.

Return Value

The **create** method returns a pointer to a **SOMDObject** which refers to a local application object.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>

Environment ev;
ReferenceData id;
InterfaceDef intfdef;
SOMDObject objref;
string fname; /* a file name to be saved with reference */
...
/* create the id for the reference */
id._maximum = id._length = strlen(fname)+1;
id._buffer = (string) SOMMalloc(strlen(fname)+1);
strcpy(id._buffer,fname);

/* get the interface def object for interface Foo*/
intfdef = _lookup_id(SOM_InterfaceRepository, &ev, "Foo");

objref = _create(SOMD_SOMOAObject,
               &ev, id, intfdef, SOMD_ImplDefObject);
...
```

Original Class

BOA

Related Information

Methods: change_id, create_constant, create_SOM_ref, dispose, get_id

deactivate_impl Method

Purpose

Indicates that a server implementation is no longer ready to process requests.

IDL Syntax

void deactivate_impl (
in ImplementationDef *impl***);**

Description

The **deactivate_impl** method indicates that the implementation is no longer ready to process requests.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
impl	A pointer to the **ImplementationDef** object representing the implementation to be deactivated.

Return Value

None.

Example

```
#include <somd.h>

ORBStatus rc;

/* server initialization code ... */

/* signal DSOM that server is ready */
_impl_is_ready(SOMD_SOMOAObject, &ev, SOMD_ImplDefObject);

for(rc = 0;rc==0;) {
    rc = _execute_next_request(SOMD_SOMOAObject, &ev, waitFlag);
    /* perform app specific code between messages here, e.g.,*/
    numMessagesProcessed++;
}

/* signal DSOM that server is deactivated */
_deactivate_impl(SOMD_SOMOAObject, &ev, SOMD_ImplDefObject);
```

Original Class

BOA

Related Information

Methods: impl_is_ready, **activate_impl_failed**, **execute_request_loop, execute_next_request**

deactivate_obj Method

Purpose

Indicates that an object server is no longer ready to process requests. *(Not implemented.)*

IDL Syntax

```
void  deactivate_obj (
                      in SOMDObject obj);
```

Description

The **deactivate_obj** method is defined by the CORBA specification, *but has a null implementation in DSOM.* This method always returns a NO_IMPLEMENT exception.

CORBA 1.1 distinguishes between servers that implement many objects ("shared"), versus servers that implement a single object ("unshared"). The **deactivate_obj** method is meant to be used by unshared servers, to indicate that the object (that is, server) is no longer ready to process requests.

DSOM does not distinguish between servers that implement a single object versus servers that implement multiple objects, so this method has no implementation.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to a **SOMDObject** which identifies the object (server) to be deactivated.

Return Value

None.

Original Class

BOA

Related Information

Methods: deactivate_impl, **impl_is_ready**, **obj_is_ready**

dispose Method

Purpose

Destroys an object reference.

IDL Syntax

void dispose (
in SOMDObject *obj***);**

Description

The **dispose** method disposes of an object reference.

Parameters

receiver	A pointer to a **BOA** object for the server.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to the object reference to be destroyed.

Return Value

None.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>

SOMDObject objref;
ReferenceData id;
InterfaceDef intfdef;
...
objref =
 _create(SOMD_SOMOAObject, &ev, id, intfdef, SOMD_ImplDefObject);
...
_dispose(SOMD_SOMOAObject, &ev, objref);
```

Original Class

BOA

Related Information

Methods: create, create_constant, create_SOM_ref, get_id

get_id Method

Purpose

Returns reference data associated with the referenced object.

IDL Syntax

ReferenceData get_id (
in SOMDObject *obj***);**

Description

The **get_id** method returns the reference data associated with the referenced object.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to a **SOMDObject** object for which to return the **ReferenceData**.

Return Value

The **get_id** method returns a **ReferenceData** structure associated with the referenced object.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>

SOMDObject objref;
ReferenceData id1, id2;
InterfaceDef intfdef;
...
objref =
 _create(SOMD_SOMOAObject,&ev, id1, intfdef, SOMD_ImplDefObject);
...
/* get the ReferenceData from a SOMDObject */
id2 = _get_id(SOMD_SOMOAObject, &ev, objref);
```

Original Class

BOA

Related Information

Methods: create, create_constant, dispose

get_principal Method

Purpose

Returns the ID of the principal that issued the request.

IDL Syntax

Principal get_principal (
in SOMDObject *obj*,
in Environment* *req_ev***);**

Description

The **get_principal** method returns the ID of the principal that issued a request.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to the object reference which is the target of the method call.
req_ev	A pointer to the **Environment** object passed as input to the request.

Return Value

The **get_principal** method returns a pointer to a **Principal** object which identifies the user and host from which a request originated.

Example

```
#include <somd.h>

/* assumed context: inside a method implementation */
void methodBody(SOMObject *somSelf, Environment *ev, ...)
{
   Principal p;
   SOMDObject selfRef;
   Environment localev;

   SOMInitEnvironment(&localev);

   /* get a reference to myself from the server object */
   selfRef =
     somdRefFromSOMObj(SOMD_ServerObject, &ev, somSelf);

   /* get principal information from the SOMOA */
   p = _get_principal(SOMD_SOMOAObject, &localev, selfRef, ev);

   printf("user = %s, host = %s\n",
         __get_userName(p), __get_hostName(p));
   ...
}
```

Original Class

BOA

Related Information

Classes: Principal

impl_is_ready Method

Purpose

Indicates that the implementation is ready to process requests.

IDL Syntax

void impl_is_ready (
in ImplementationDef *impl***);**

Description

The **impl_is_ready** method Indicates that the implementation is ready to process requests.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
impl	A pointer to the **ImplementationDef** object indicating which implementation is ready.

Return Value

None.

Example

```
#include <somd.h> /* needed by all servers */

main(int argc, char **argv)
{
   Environment ev;
   SOM_InitEnvironment(&ev);

   /* Initialize the DSOM run-time environment */
   SOMD_Init(&ev);

   /* Retrieve its ImplementationDef from the Implementation
      Repository by passing its implementation ID as a key */
   SOMD_ImplDefObject =
      _find_impldef(SOMD_ImplRepObject, &ev, argv[1]);

   /* Tell DSOM that the server is ready to process requests */
   _impl_is_ready(SOMD_SOMOAObject, &ev, SOMD_ImplDefObject);
   ...
}
```

Original Class

BOA

Related Information

Methods: deactivate_impl, **activate_impl_failed**, **obj_is_ready**, **execute_request_loop, execute_next_request**

obj_is_ready Method

Purpose

Indicates that an object (server) is ready to process requests. *(Not implemented.)*

IDL Syntax

void obj_is_ready (
in SOMDObject *obj***,**
in ImplementationDef *impl***);**

Description

The **obj_is_ready** method is defined by the CORBA specification, *but has a null implementation in DSOM.* This method always returns a NO_IMPLEMENT exception.

CORBA 1.1 distinguishes between servers that implement many objects (“shared”), versus servers that implement a single object (“unshared”). The **obj_is_ready** method is meant to be used by unshared servers, to indicate that the object (that is, server) is ready to process requests.

DSOM does not distinguish between servers that implement a single object versus servers that implement multiple objects, so this method has no implementation.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to a **SOMDObject** which identifies the object (server) that is ready.
impl	A pointer to the **ImplementationDef** object representing the object that is ready.

Return Value

None.

Original Class

BOA

Related Information

Methods: impl_is_ready, **deactivate_impl**, **deactivate_obj**, **activate_impl_failed**

set_exception Method

Purpose

Returns an exception to a client.

IDL Syntax

void set_exception (
in exception_type *major*,
in string *except_name*,
in void* *param***);**

Description

The **set_exception** method returns an exception to the client. The *major* parameter can have one of three possible values:

NO_EXCEPTION — indicates a normal outcome of the operation. It is not necessary to invoke **set_exception** to indicate a normal outcome; it is the default behavior if the method simply returns.

USER_EXCEPTION — indicates a user-defined exception.

SYSTEM_EXCEPTION — indicates a system-defined exception.

Parameters

receiver	A pointer to a **BOA** (**SOMOA**) object for the server.
env	A pointer to the **Environment** structure for the method caller.
major	One of the exception types NO_EXCEPTION, USER_EXCEPTION, or SYSTEM_EXCEPTION.
except_name	A **string** representing the exception type identifier.
param	A pointer to the associated data.

Return Value

None.

Example

```
#include <somd.h>
#include <myobject.h>   /* provided by user */

/* assuming following IDL declarations in the MyObject interface:
 *    exception foo;
 *    void myMethod() raises (BadCall);
 * then within the implementation of myMethod, the
 * following call can raise a BadCall exception: */

_set_exception(SOMD_SOMOAObject,
               &ev, USER_EXCEPTION, ex_MyObject_BadCall, NULL);
```

Original Class

BOA

Context Class

Description

The **Context** class implements the CORBA Context object described in section 6.5 beginning on page 116 of CORBA 1.1. A **Context** object contains a list of properties, each consisting of a name and a string value associated with that name. **Context** objects are created/accessed by the **get_default_context** method defined in the **ORB** object.

File Stem

cntxt

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

create_child
delete_values
destroy*
get_values
set_one_value
set_values

(* The **destroy** method was defined as **delete** in CORBA 1.1, which conflicts with the **delete** operator in C++. However, there is a **Context_delete** macro defined for CORBA compatibility.)

Overridden Methods

somInit

create_child Method

Purpose

Creates a child of a **Context** object.

IDL Syntax

ORBStatus create_child (
in Identifier *ctx_name*,
out Context *child_ctx*);

Description

The **create_child** method creates a child **Context** object.

The returned **Context** object is chained to its parent. That is, searches on the child **Context** object will look in the parent (and so on, up the **Context** tree), if necessary, for matching property names.

Parameters

receiver	A pointer to the **Context** object for which a child is to be created.
env	A pointer to the **Environment** structure for the method caller.
ctx_name	The name of the child **Context** to be created.
child_ctx	The address where a pointer to the created child **Context** object is to be stored.

Return Value

The **create_child** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
Context cxt, newcxt;
long rc;
...
/* get the process' default Context */
rc = _get_default_context(SOMD_ORBObject, &ev, &cxt);
/* make newcxt a child Context of the default Context (cxt) */
rc = _create_child(cxt, &ev, "myContext", &newcxt);
```

Original Class

Context

delete_values Method

Purpose

Deletes property value(s).

IDL Syntax

ORBStatus delete_values (
in Identifier *prop_name***);**

Description

The **delete_values** method deletes the specified property value(s) from a **Context** object. If *prop_name* has a trailing wildcard character(“*”), then all property names that match will be deleted.

Search scope is always limited to the specified **Context** object.

If no matching property is found, an exception is returned.

Parameters

receiver	A pointer to the **Context** object from which values will be deleted.
env	A pointer to the **Environment** structure for the method caller.
prop_name	An identifier specifying the property value(s) to be deleted.

Return Value

The **delete_values** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
Context cxt, newcxt;
long rc;
...
/* get the process’ default Context */
rc = _get_default_context(SOMD_ORBObject, &ev, &cxt);
/* make newcxt a child Context of the default Context (cxt) */
rc = _create_child(cxt, &ev, ”myContext”, &newcxt);
rc = _set_one_value(newcxt, &ev, ”username”, ”joe”);
...
rc = _delete_values(newcxt, &ev, ”username”);
```

Original Class

Context

Related Information

Methods: set_one_value, set_values, get_values

destroy Method (for a Context object)

Purpose

Deletes a **Context** object.

IDL Syntax

ORBStatus destroy (
in Flags *del_flag***);**

Description

The **destroy** method deletes the specified **Context** object.

NOTE: This method is called “delete” in the CORBA 1.1 specification. However, the word “delete” is a reserved operator in C++, so the name “destroy” was chosen as an alternative. For CORBA compatibility, a macro defining **Context_delete** as an alias for **destroy** has been included in the C header files.

Parameters

receiver	A pointer to the **Context** object to be deleted.
env	A pointer to the **Environment** structure for the method caller.
del_flag	A bitmask (unsigned long). If the option flag CTX_DELETE_DESCENDENTS is specified, the method deletes the indicated **Context** object and all of its descendent **Context** objects. Or, a zero value indicates the flag is not set.

Return Value

The **destroy** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
Context cxt, newcxt;
long rc;
...
/* get the process' default Context */
rc = _get_default_context(SOMD_ORBObject, &ev, &cxt);
/* make newcxt a child Context of the default Context (cxt) */
rc = _create_child(cxt, &ev, "myContext", &newcxt);
...
/* assuming no descendent Contexts have been
 * created from newcxt, we can destroy newcxt with flags=0
 */
rc = _destroy(newcxt, &ev, (Flags) 0);
```

Original Class

Context

get_values Method

Purpose

Retrieves the specified property values.

IDL Syntax

ORBStatus get_values (
in Identifier *start_scope***,**
in Flags *op_flags***,**
in Identifier *prop_name***,**
out NVList *values***);**

Description

The **get_values** method retrieves the specified **Context** property values(s). If *prop_name* has a trailing wildcard character("*"), then all matching properties and their values are returned. OWNERSHIP of the returned **NVList** object is transferred to the caller.

If no properties are found, an error is returned and no property list is returned.

Scope indicates the level at which to initiate the search for the specified properties. If a property is not found at the indicated level, the search continues up the **Context** object tree until a match is found or all **Context** objects in the chain have been exhausted.

If scope name is omitted, the search begins with the specified **Context** object. If the specified scope name is not found, an exception is returned.

Parameters

receiver	A pointer to the **Context** object from which the properties are to be retrieved.
env	A pointer to the **Environment** structure for the method caller.
start_scope	An **Identifier** specifying the name of the **Context** object at which search for the properties should commence.
op_flags	This parameter is currently not used (the CTX_RESTRICT_SCOPE flag is currently not supported); callers can simply pass zero.
prop_name	An **Identifier** specifying the name of the property value(s) to return.
values	The address to store a pointer to the resulting **NVList** object.

Return Value

The **get_values** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
Context cxt1, cxt2;
string *cxt1props;
long rc, i, numprops;
NVList nvp;
...
for (i= numprops; i > 0; i--) {
 /* get the value of the *cxt1props property from cxt1 */
 rc = _get_values(cxt1, &ev, NULL, (Flags) 0, *cxt1props, &nvp);
 /* and if found then update cxt2 with that name-value pair */
 if (rc == 0) rc = _set_values(cxt2, &ev, nvp);
 _free(nvp,&ev);
 cxt1props++;
}
```

Original Class

Context

Related Information

Methods: set_one_value, set_values, delete_values

set_one_value Method

Purpose

Adds a single property to the specified **Context** object.

IDL Syntax

ORBStatus set_one_value (
in Identifier *prop_name***,**
in string *value***);**

Description

The **set_one_value** method adds a single property to the specified **Context** object.

Parameters

receiver A pointer to the **Context** object to which the value is to be added.

env A pointer to the **Environment** structure for the method caller.

prop_name The name of the property to be added. The *prop_name* should not end in an asterisk (*).

value The value of the property to be added.

Return Value

The **set_one_value** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
Context cxt, newcxt;
long rc;
...
/* get the process' default Context */
rc = _get_default_context(SOMD_ORBObject, &ev, &cxt);
/* make newcxt a child Context of the default Context (cxt) */
rc = _create_child(cxt, &ev, "myContext", &newcxt);
rc = _set_one_value(newcxt, &ev, "username", "joe");
```

Original Class

Context

Related Information

Methods: set_values, get_values, delete_values

set_values Method

Purpose

Adds/changes one or more property values in the specified **Context** object.

IDL Syntax

ORBStatus set_values (
in NVList *values***);**

Description

The **set_values** method sets one or more property values in the specified **Context** object. In the **NVList**, the flags field must be set to zero, and the **TypeCode** field associated with an attribute value must be **TC_string**.

Parameters

receiver	A pointer to the **Context** object for which the properties are to be set.
env	A pointer to the **Environment** structure for the method caller.
values	A pointer to an **NVList** object containing the properties to be set. The property names in the **NVList** should not end in an asterisk (`*`).

Return Value

The **set_values** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
Context cxt1, cxt2;
string *cxt1props;
long rc, i, numprops;
NVList nvp;
...
for (i= numprops; i > 0; i--) {
 /* get the value of the *cxt1props property from cxt1 */
 rc = _get_values(cxt1, &ev, NULL, (Flags) 0, *cxt1props, &nvp);
 /* and if found then update cxt2 with that name-value pair */
 if (rc == 0) rc = _set_values(cxt2, &ev, nvp);
 _free(nvp,&ev);
 cxt1props++;
}
```

Original Class

Context

Related Information

Methods: set_one_value, get_values, delete_values

ImplementationDef Class

Description

The **ImplementationDef** class defines attributes necessary for the DSOM daemon to find and activate the implementation of an object.

*** Note:** Details of the **ImplementationDef** object are not currently defined in the CORBA 1.1 specification; the attributes which have been defined are required by DSOM.

File Stem

impldef

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

impl_id (string)

Contains the DSOM-generated identifier for a server implementation.

impl_alias (string)

Contains the "alias" (user-friendly name) for a server implementation.

impl_program (string)

Contains the name of the program or command file which will be executed when a process for this server is started automatically by **somdd**. If the full pathname is not specified, the directories specified in the PATH environment variable will be searched for the named program or command file.

Optionally, the server program can be run under control of a "shell" or debugger, by specifying the shell or debugger name first, followed by the name of the server program. (A space separates the two program names.) For example,

```
dbx myserver
```

Servers that are started automatically by **somdd** will always be passed their **impl_id** as the first parameter.

impl_flags (Flags)

Contains a bit-vector of flags used to identify server options. Currently, the IMPLDEF_MULTI_THREAD flag indicates that each request should be executed on a separate thread (OS/2 only). IMPLDEF_DISABLE_SVR indicates that the server process has been disabled from starting.

impl_server_class (string)

Contains the name of the **SOMDServer** class or subclass created by the server process.

impl_refdata_file (string)

Contains the full pathname of the file used to store **ReferenceData** for the server.

impl_refdata_bkup (string)

Contains the full pathname of the backup mirror file used to store **ReferenceData** for the server.

impl_hostname (string)

Contains the hostname of the machine where the server is located.

Notes

Currently, when stored in the Implementation Repository, file names used in **ImplementationDef**s are limited to 255 bytes. Implementations aliases used in **ImplementationDef**s are limited to 50 bytes. Class names used in **ImplementationDef**s are limited to 50 bytes. Hostnames are limited to 32 bytes.

ImplRepository Class

Description

The **ImplRepository** class defines operations necessary to query and update the DSOM Implementation Repository.

Note: The Implementation Repository is described in concept in the CORBA 1.1 specification, but no standard interfaces have been defined. These interfaces have all been introduced by DSOM. In addition to using the following interfaces, the DSOM Implementation Repository can be queried and updated using the **regimpl** tool.

File Stem

implrep

Base

SOMObject

Metaclass

SOMMSingleInstance

Ancestor Classes

SOMObject

New Methods

add_class_to_impldef
add_impldef
delete_impldef
find_all_impldefs
find_classes_by_impldef
find_impldef
find_impldef_by_alias
find_impldef_by_class
remove_class_from_all
remove_class_from_impldef
update_impldef

Overridden Methods

somInit
somUninit

add_class_to_impldef Method

Purpose

Associates a class with a server.

IDL Syntax

void add_class_to_impldef (
in ImplId *implid,*
in string *classname* **);**

Description

Associates a class, identified by name, with a server, identified by its **ImplId**. This type of association is used to lookup server implementations via the **find_impldef_by_class** method.

Parameters

receiver	A pointer to the **ImplRepository** object.
env	A pointer to the **Environment** structure for the method caller.
implid	The **ImplId** identifier for the **ImplementationDef** of the desired server.
classname	A **string** identifying the class name.

Return Value

An exception is returned if there was an error updating the Implementation Repository.

Example

```
#include <somd.h>

Environment ev;
SOMDServer server;
ImplementationDef impldef;
ImplId implid;
...
server = _somdFindServerByName(SOMD_ObjectMgr,&ev,"stackServer");
impldef = _get_implementation(server,&ev);
implid = __get_impl_id(impldef,&ev);
_add_class_to_impldef(SOMD_ImplRepObject,&ev,implid,"Queue");
```

Original Class

ImplRepository

add_impldef Method

Purpose

Adds an implementation definition to the Implementation Repository.

IDL Syntax

void add_impldef (
in ImplementationDef *impldef***);**

Description

Adds the specified **ImplementationDef** object to the Implementation Repository.

Note: the **impl_id** field of the **ImplementationDef** is ignored. A new **impl_id** value will be created for the newly added **ImplementationDef.**

Parameters

receiver	A pointer to the **ImplRepository** object.
env	A pointer to the **Environment** structure for the method caller.
impldef	A pointer to the **ImplementationDef** object to add to the Implementation Repository.

Return Value

An exception is returned if there was an error updating the Implementation Repository.

Example

```
#include <somd.h>

Environment ev;
ImplementationDef impldef;
...
impldef = ImplementationDefNew();
__set_impl_program(impldef,&ev,"/u/servers/myserver");
/* set more of the impldef's attributes here */
...
_add_impldef(SOMD_ImplRepObject,&ev,impldef);
```

Original Class

ImplRepository

delete_impldef Method

Purpose

Deletes an implementation definition from the Implementation Repository.

IDL Syntax

void delete_impldef (
in ImplId *implid* **);**

Description

Deletes the specified **ImplementationDef** object from the Implementation Repository.

Parameters

receiver	A pointer to the **ImplRepository** object.
env	A pointer to the **Environment** structure for the method caller.
implid	The **ImplId** that identifies the server implementation of interest.

Return Value

An exception is returned if there was an error updating the Implementation Repository.

Example

```
#include <somd.h>

Environment ev;
ImplementationDef impldef;
...
impldef =
   _find_impldef_by_name(SOMD_ImplRepObject,&ev,"stackServer");
_delete_impldef(SOMD_ImplRepObject,&ev,__get_impl_id(impldef,&ev));
```

Original Class

ImplRepository

find_all_impldefs Method

Purpose

Returns all the implementation definitions in the Implementation Repository.

IDL Syntax

ORBStatus find_all_impldefs (out sequence<ImplementationDef> *outimpldefs***);**

Description

The **find_all_impldefs** method searches the Implementation Repository and returns all the **ImplementationDef** objects in it.

Parameters

receiver	A pointer to an object of class **ImplRepository**.
ev	A pointer to the **Environment** structure for the calling method.
outimpldefs	A sequence of **ImplementationDefs** is returned.

Return Value

A zero is returned to indicate success; otherwise, a DSOM error code is returned.

Example

```
#include <somd.h>

Environment ev;
sequence (ImplementationDef) impldefs;

. . .

find_all_impldefs(SOMD_ImplRepObject, &ev, &impldefs);
```

Original Class

ImplRepository

find_classes_by_impldef Method

Purpose

Returns a sequence of class names associated with a server.

IDL Syntax

sequence<string> find_classes_by_impldef (
in ImplId *implid* **);**

Description

The **find_classes_by_impldef** method searches the class index and returns the sequence of class names supported by a server with the specified *implid*.

Parameters

receiver	A pointer to the **ImplRepository** object.
env	A pointer to the **Environment** structure for the method caller.
implid	The **ImplId** that identifies the server implementation of interest.

Return Value

A sequence of strings is returned. *Ownership* of the sequence structure, the string array buffer, and the strings themselves is transferred to the caller.

An exception is returned if there was an error reading the Implementation Repository.

Example

```
#include <somd.h>

Environment ev;
SOMDServer server;
ImplementationDef impldef;
ImplId implid;
sequence(string) classes;
...
server = _find_server_by_name(SOMD_ObjectMgr,&ev,"stackServer");
impldef = _get_implementation(server,&ev);
implid = __get_impl_id(impldef,&ev);
classes = _find_classes_by_impldef(SOMD_ImplRepObject,&ev,implid);
```

Original Class

ImplRepository

find_impldef Method

Purpose

Returns a server implementation definition given its ID.

IDL Syntax

ImplementationDef find_impldef (
in ImplId *implid***);**

Description

Finds and returns the **ImplementationDef** object whose ID is *implid*.

Parameters

receiver	A pointer to the **ImplRepository** object.
env	A pointer to the **Environment** structure for the method caller.
implid	The **ImplId** of the desired **ImplementationDef.**

Return Value

A copy of the desired **ImplementationDef** object is returned. *Ownership* of the object is transferred to the caller.

An exception is returned if there was an error reading the Implementation Repository.

Example

```
#include <somd.h>

main(int argc, char **argv)
{
   Environment ev;
   SOM_InitEnvironment(&ev);

   /* Initialize the DSOM run-time environment */
   SOMD_Init(&ev);

   /* Retrieve its ImplementationDef from the Implementation
      Repository by passing its implementation ID as a key */
   SOMD_ImplDefObject =
      _find_impldef(SOMD_ImplRepObject, &ev, argv[1]);

   /* Tell DSOM that the server is ready to process requests */
   _impl_is_ready(SOMD_SOMOAObject, &ev, SOMD_ImplDefObject);
   ...
}
```

Original Class

ImplRepository

find_impldef_by_alias Method

Purpose

Returns a server implementation definition, given its user-friendly alias.

IDL Syntax

ImplementationDef find_impldef_by_alias (
in string *alias_name***);**

Description

Finds and returns the **ImplementationDef** object whose alias is *alias_name.*

Parameters

receiver	A pointer to the **ImplRepository** object.
env	A pointer to the **Environment** structure for the method caller.
alias_name	User-friendly name used to identify the implementation.

Return Value

A copy of the desired **ImplementationDef** object is returned, and *ownership* of the object is transferred to the caller. Or, if the specified alias is not found in the Implementation Repository, NULL is returned.

An exception is returned if there was an error reading the Implementation Repository.

Example

```
#include <somd.h>

Environment ev;
ImplementationDef impldef;
...
impldef =
   _find_impldef_by_alias(SOMD_ImplRepObject,&ev,"stackServer");
_delete_impldef(SOMD_ImplRepObject,&ev,__get_impl_id(impldef,&ev));
```

Original Class

ImplRepository

find_impldef_by_class Method

Purpose

Returns a sequence of implementation definitions for servers that are associated with a specified class.

IDL Syntax

sequence<ImplementationDef> find_impldef_by_class (
in string *classname***);**

Description

Returns a sequence of **ImplementationDefs** for those servers that have registered an association with a specified class. Typically, a server will be associated with the classes it knows how to implement by registering its known classes via the **add_class_to_impldef** method.

Parameters

receiver	A pointer to the **ImplRepository** object.
env	A pointer to the **Environment** structure for the method caller.
classname	A **string** whose value is the class name of interest.

Return Value

Copies of all **ImplementationDef** objects are returned in a sequence. *Ownership* of the sequence structure, the object array buffer, and the objects themselves is transferred to the caller.

An exception is returned if there was an error reading the Implementation Repository.

Example

```
#include <somd.h>

Environment ev;
sequence(ImplementationDef) impldefs;
...
impldefs =
   _find_impldef_by_class(SOMD_ImplRepObject,&ev,"Stack");
```

Original Class

ImplRepository

remove_class_from_all Method

Purpose

Removes the association of a particular class from all servers.

IDL Syntax

void remove_class_from_all (in string *className***);**

Description

The **remove_class_from_all** method removes the *className* from all of the **ImplementationDefs**.

Parameters

receiver	A pointer to an object of class **ImplRepository**.
ev	A pointer to the **Environment** structure for the calling method.
className	A string whose value is the class name of interest.

Return Value

None.

Example

```
#include <somd.h>

Environment ev;
...
remove_class_from_all(SOMD_ImplRepObject, &ev, "Stack");
```

Original Class

ImplRepository

remove_class_from_impldef Method

Purpose

Removes the association of a particular class with a server.

IDL Syntax

void remove_class_from_impldef (
in ImplId *implid,*
in string *classname* **);**

Description

Removes the specified class name from the set of class names associated with the server implementation identified by *implid.*

Parameters

receiver	A pointer to the **ImplRepository** object.
env	A pointer to the **Environment** structure for the method caller.
implid	A pointer to an **ImplRepository** object.
classname	A **string** whose value is the class name of interest.

Return Value

An exception is returned if there was an error updating the Implementation Repository.

Example

```
#include <somd.h>

Environment ev;
SOMDServer server;
ImplementationDef impldef;
ImplId implid;
...
server = _find_server_by_name(SOMD_ObjectMgr,&ev,"stackServer");
impldef = _get_implementation(server,&ev);
implid = __get_impl_id(impldef,&ev);
_remove_class_from_impldef(SOMD_ImplRepObject,
                           &ev,implid,"Queue");
```

Original Class

ImplRepository

update_impldef Method

Purpose

Updates an implementation definition in the Implementation Repository.

IDL Syntax

void update_impldef (
in ImplementationDef *impldef***);**

Description

Replaces the state of the specified **ImplementationDef** object in the Implementation Repository. The ID of the *impldef* determines which object gets updated in the Implementation Repository.

Parameters

receiver A pointer to the **ImplRepository** object.

env A pointer to the **Environment** structure for the method caller.

impldef A pointer to an **ImplementationDef** object, whose values are to be saved in the Implementation Repository.

Return Value

An exception is returned if there was an error updating the Implementation Repository.

Example

```
#include <somd.h>

Environment ev;
SOMDObject objref;
ImplementationDef impldef;
...
impldef = _get_implementation(objref,&ev);
__set_impl_program(impldef,&ev,"/u/joe/bin/myserver");
_update_impldef(SOMD_ImplRepObject,&ev,impldef);
```

Original Class

ImplRepository

NVList Class

Description

The type **NamedValue** is a standard datatype defined in CORBA (see the CORBA 1.1 page 106). It can be used either as a parameter type or as a mechanism for describing arguments to a request. The **NVList** class implements the **NVList** object used for constructing lists composed of **NamedValues**. **NVLists** can be used to describe arguments passed to request operations or to pass lists of property names and values to context object routines. Additional information about **NVList** is contained in Chapter 6 of the CORBA 1.1 specification.

File Stem

nvlist

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

add_item
free
free_memory
get_count
get_item*
set_item*

(* These methods were added by DSOM to supplement
the published CORBA 1.1 methods.)

Overridden Methods

somInit

add_item Method

Purpose

Adds an item to the specified **NVList**.

IDL Syntax

ORBStatus add_item (
in Identifier *item_name*,
in TypeCode *item_type*,
in void* *value*,
in long *value_len*,
in Flags *item_flags***);**

Description

The **add_item** method adds an item to the end of the specified list.

Parameters

receiver	A pointer to the **NVList** object to which the item will be added.
env	A pointer to the **Environment** structure for the method caller.
item_name	The name of the item to be added.
item_type	The data type of the item to be added.
value	A pointer to the value of the item to be added.
value_len	The length of the item value to be added.
item_flags	A **Flags** bitmask (unsigned long). The *item_flags* can be one of the following values to indicate parameter direction:

ARG_IN	The argument is input only.
ARG_OUT	The argument is output only.
ARG_INOUT	The argument is input/output.

In addition, item_flags may also contain the following values:

IN_COPY_VALUE	An internal copy of the argument is made and used.
DEPENDENT_LIST	Indicates that a specified sublist must be freed when the parent list is freed.

Return Value

The **add_item** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
NVList plist;
ORBStatus rc;
...
rc = _create_list(SOMD_ORBObject, &ev, 0, &plist);
rc = _add_item(plist, &ev, ”firstname”, TC_string, ”Joe”, 3, 0);
rc = _add_item(plist, &ev, ”lastname”, TC_string, ”Schmoe”, 5, 0);
```

Original Class

NVList

Related Information

Methods: free, **free_memory**, **get_count, get_item, set_item, create_list**

free Method

Purpose

Frees a specified **NVList.**

IDL Syntax

ORBStatus free ( );

Description

The **free** method frees a n **NVList** object and any associated memory. It makes an implicit call to the **free_memory** method.

Parameters

receiver	A pointer to the **NVList** object to be freed.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
long nargs;
NVList arglist;
ORBStatus rc;
...
rc = _create_list(SOMD_ORBObject, &ev, nargs, &arglist);
...
rc= _free(arglist,&ev);
```

Original Class

NVList

Related Information

Functions: ORBfree
Methods: free_memory

free_memory Method

Purpose

Frees any dynamically allocated out-arg memory associated with the specified list.

IDL Syntax

ORBStatus free_memory ( );

Description

The **free_memory** method frees any dynamically allocated out-arg memory associated with the specified list, without freeing the list object itself. This would be useful when invoking a DII request multiple times with the same **NVList.**

Parameters

receiver A pointer to the **NVList** object whose out-arg memory is to be freed.

env A pointer to the **Environment** structure for the method caller.

Return Value

The **free_memory** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code repeatedly invokes a request:
 *      result = methodLong(fooObj, &ev, 100, 200);
 * using the DII.
 */

Environment ev;
NVList arglist;
NamedValue result;
long rc;
Foo fooObj;
Request reqObj;

/* See example code for "invoke" to see how the argList is built */

/* Create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, "methodLong",
                     arglist, &result, &reqObj, (Flags)0);

/* Repeatedly invoke the Request */
for (;;) {
   rc = _invoke(reqObj, &ev, (Flags)0);
   ...
   rc= _free_memory(arglist,&ev);  /* free out args */
}
...
```

Original Class

NVList

Related Information

Functions: ORBfree
Methods: free

get_count Method

Purpose

Returns the total number of items allocated for a list.

IDL Syntax

ORBStatus get_count (
out long *count***);**

Description

The **get_count** method returns the total number of allocated items in the specified list.

Parameters

receiver	A pointer to the **NVList** object on which count is desired.
env	A pointer to the **Environment** structure for the method caller.
count	A pointer to where the method will store the **long** integer count value.

Return Value

The **get_count** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>

Environment ev;
long nargs, list_size;
NVList arglist;
ORBStatus rc;
...
rc = _create_list(SOMD_ORBObject, &ev, nargs, &arglist);
...
rc = _get_count(arglist,&ev,&list_size);
```

Original Class

NVList

Related Information

Methods: add_item, **get_item, set_item, create_list**

get_item Method

Purpose

Returns the contents of a specified list item.

IDL Syntax

ORBStatus get_item (
in long *item_number*,
out Identifier *item_name*,
out TypeCode *item_type*,
out void* *value*,
out long *value_len*,
out Flags *item_flags***);**

Description

The **get_item** method gets an item from the specified list. Items are numbered from 0 through *N*. The mode flags can be one of the following values:

The **get_item** method transfers ownership of storage allocated for the item value to the caller.

Parameters

receiver	A pointer to an **NVList** object.
env	A pointer to the **Environment** structure for the method caller.
item_number	The position (index) of the item in the list. The item_number ranges from 0 to *n*–1, where *n* is the total number of items in the list.
item_name	A pointer to where the name of the item should be returned.
item_type	A pointer to where the data type of the item should be returned.
value	A pointer to where a pointer to the value of the item should be returned.
value_len	A pointer to where the length of the item value should be returned.
item_flags	A **Flags** bitmask (unsigned long). The *item_flags* can be one of the following values indicating parameter direction.

ARG_IN	The argument is input only.
ARG_OUT	The argument is output only.
ARG_INOUT	The argument is input/output.

In addition, *item_flags* can have the following values:

IN_COPY_VALUE	Indicates a copy of the argument is contained and used by the **NVList**.
DEPENDENT_LIST	Indicates that a specified sublist must be freed when the parent list is freed.

Return Value

The **get_item** method returns 0 for success, or a DSOM error code for failure (often because item_number+1 exceeds the number of items in the list).

Example

```
#include <somd.h>

Environment ev;
long i, nArgs;
ORBStatus rc;
Identifier name;
TypeCode typeCode;
void *value;
long len;
Flags flags;
NVList argList;
...
/* get number of args */
rc = _get_count(argList, ev, &nArgs);
for (i = 0; i < nArgs; i++) {
   /* get item description */
   rc = _get_item(argList,
                  &ev,
                  i,
                  &name,
                  &typeCode,
                  &value,
                  &len,
                  &flags);
   ...
   }
```

Original Class

NVList

Related Information

Methods: add_item, set_item, create_list

set_item Method

Purpose

Sets the contents of an item in a list.

IDL Syntax

ORBStatus set_item (
in long *item_number*,
in Identifier *item_name*,
in TypeCode *item_type*,
in void* *value*,
in long *value_len*,
in Flags *item_flags***);**

Description

The **set_item** method sets the contents of an item in the list.

Parameters

receiver A pointer to an **NVList** which contains the item to be set.

env A pointer to the **Environment** structure for the method caller.

item_number The position (index) of the item in the list. The *item_number* ranges from 0 to *n*–1, where *n* is the total number of items in the list.

item_name The name of the set item.

item_type The data type of the set item.

value A pointer to the value of the set item.

value_len The length of the set item value.

item_flags A **Flags** bitmask (unsigned long). The *item_flags* can be one of the following values to indicate parameter direction:

ARG_IN	The argument is input only.
ARG_OUT	The argument is output only.
ARG_INOUT	The argument is input/output.

In addition, *item_flags* may also contain the following values:

IN_COPY_VALUE	Indicates an internal copy of the argument is made and used.
DEPENDENT_LIST	Indicates that a specified sublist must be freed when the parent list is freed.

Return Value

The **set_item** method returns 0 on successful completion or a DSOM error code upon failure (often because item_number+1 exceeds the number of items in the list).

Example

```
#include <somd.h>

Environment ev;
long i, nArgs;
ORBStatus rc;
Identifier name;
TypeCode typeCode;
void *value;
long len;
Flags flags;
NVList argList;
...
/* get number of args */
rc = _get_count(argList, ev, &nArgs);
for (i = 0; i < nArgs; i++) {
   /* change item description */
   rc = _set_item(argList,
                  &ev,
                  i,
                  name,
                  typeCode,
                  value,
                  len,
                  flags);
   ...
   }
```

Original Class

NVList

Related Information

Methods: get_item, add_item, create_list

ObjectMgr Class

Description

The **ObjectMgr** class provides a uniform, universal abstraction for any sort of object manager. Object Request Brokers, persistent storage managers, and OODBMSs are examples of object managers.

This is an abstract base class, which defines the “core” interface for an object manager. It provides basic methods that:

- Create a new object of a certain class,
- Return a (persistent) ID for an object,
- Return a reference to an object associated with an ID,
- Free an object (that is, release any local memory associated with the object without necessarily destroying the object itself), or
- Destroy an object.

Note: The **ObjectMgr** is an *abstract* class and should not be instantiated. Any subclass of **ObjectMgr** must provide implementations for all **ObjectMgr** methods. In DSOM, the class **SOMDObjectMgr** provides a DSOM-specific implementation.

File Stem

om

Base

SOMObject

Metaclass

SOMMSingleInstance

Ancestor Classes

SOMObject

Subclasses

SOMDObjectMgr

New Methods

somdDestroyObject*
somdGetIdFromObject*
somdGetObjectFromId*
somdNewObject*
somdReleaseObject*

(***** This class and its methods were added by DSOM to supplement the published CORBA 1.1 interfaces.)

somdDestroyObject Method

Purpose

Requests destruction of the target object.

IDL Syntax

void somdDestroyObject (
in SOMObject *obj***);**

Description

The **somdDestroyObject** method indicates that the object manager should destroy the specified object. Storage associated with the object is freed.

In DSOM, the **SOMDObjectMgr** forwards the deletion request to the remote server, and then frees the local proxy object.

Parameters

receiver	A pointer to an **ObjectMgr** object.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to the object to be freed.

Return Value

None.

Example

```
#include <somd.h>

Stack stk;
Environment ev;
SOMDServer server;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
server =
  _somdFindAnyServerByClass(SOMD_ObjectMgr, &ev,"Stack");
stk = _somdCreateObj(server, &ev, "Stack", "");
...
_somdDestroyObject(SOMD_ObjectMgr, &ev, stk);
```

Original Class

ObjectMgr

Related Information

Methods: somdReleaseObject, somdCreateObj, somdTargetFree, release

somdGetIdFromObject Method

Purpose

Returns an ID for an object managed by a specified Object Manager.

IDL Syntax

string somdGetIdFromObject (
in SOMObject *obj***);**

Description

The **somdGetIdFromObject** method returns the persistent ID for an object managed by the specified Object Manager. This ID is unambiguous — it always refers to the same object.

The **somdGetIdFromObject** method transfers *ownership* of storage allocated for the string to the caller. The caller should free the result using the **ORBfree** function, rather than the **SOMFree** function.

Parameters

receiver	A pointer to an **ObjectMgr** object.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to the **SOMDObject** object for which an ID is needed.

Return Value

The **somdGetIdFromObject** method returns a string representing the ID of the specified object.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
string somdObjectId;
/*note that ”SOMDObject Identifiers” are just strings */

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);

/* create a remote Car object */
car = _somdNewObject(SOMD_ObjectMgr, &ev, “Car”, ””);

/* save the reference to the object */
somdObjectId = _somdGetIdFromObject(SOMD_ObjectMgr, &ev, car);
FileWrite(”/u/joe/mycar”, somdObjectId);
...
```

Original Class

ObjectMgr

Related Information

Methods: somdGetObjectFromId

somdGetObjectFromId Method

Purpose

Finds and activates an object implemented by a specified object manager, given its ID.

IDL Syntax

SOMObject somdGetObjectFromId (
in string *id***);**

Description

The **somdGetObjectFromId** method finds and activates an object implemented by this object manager, given its ID.

The **somdGetObjectFromId** method transfers *ownership* to the caller.

Parameters

receiver	A pointer to an **ObjectMgr** object.
env	A pointer to the **Environment** structure for the method caller.
id	A string representing an object ID.

Return Value

The **somdGetObjectFromId** method returns a pointer to the object with the specified ID.

Example

```
#include <somd.h>
#include <car.h>
Environment ev;
Car car;
string somdObjectId;
...
/* restore proxy from its string form */
FileRead(”/u/joe/mycar”, &somdObjectId);
car = _somdGetObjectFromId(SOMD_ObjectMgr, &ev, somdObjectId);
...
```

Original Class

ObjectMgr

Related Information

Methods: somdGetIdFromObject

somdNewObject Method

Purpose

Returns a new object of the named class.

IDL Syntax

SOMObject somdNewObject (
in Identifier *objclass***,**
in string *hints***);**

Description

The **somdNewObject** method returns a new object of the class specified by *objclass*. Application-specific creation options can be supplied via the *hints* parameter.

In DSOM, the **SOMDObjectMgr** selects a random server which has advertised knowledge of the desired class *objclass*, and forwards the creation request to that server. The *hints* field is currently ignored by the **SOMDObjectMgr**.

Parameters

receiver	A pointer to an **ObjectMgr** object.
env	A pointer to the **Environment** structure for the method caller.
objclass	An **Identifier** representing the type of the new object.
hints	A **string** which may optionally be used to specify special creation options.

Return Value

The **somdNewObject** method returns a **SOMObject.** *Ownership* of the new object is transferred to the caller.

Example

```
#include <somd.h>
#include <stack.h>   /* provided by user */

Stack stk;
Environment ev;
SOMDServer server;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
stk = _somdNewObject(SOMD_ObjectMgr, &ev, "Stack", "");
...
_somdDestroyObject(SOMD_ObjectMgr, &ev, stk);
```

Original Class

ObjectMgr

Related Information

Methods: somdDestroyObject, somdReleaseObject

somdReleaseObject Method

Purpose

Indicates that the client has finished using the object.

IDL Syntax

void somdReleaseObject (
in SOMObject *obj***);**

Description

The **somdReleaseObject** method indicates that the client has finished using the specified object. This allows the object manager to free the bookkeeping information associated with the object, if any. The object may also be passivated, but it is not destroyed.

In DSOM, **somdReleaseObject** causes the client's proxy for the target object of interest to be freed; the target object is not freed.

Parameters

receiver	A pointer to an **ObjectMgr** object.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to the object to be released.

Return Value

None.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
string somdObjectId;
...
/* restore proxy from its string form */
FileRead("/u/joe/mycar", &somdObjectId);
car = _somdGetObjectFromId(SOMD_ObjectMgr, &ev, somdObjectId);
...
_somdReleaseObject(SOMD_ObjectMgr, &ev, car);
```

Original Class

ObjectMgr

Related Information

Methods: somdDestroyObject, somdNewObject, somdTargetFree, release

ORB Class

Description

The **ORB** class implements the CORBA ORB object described in Chapter 8 of the CORBA 1.1 specification. The **ORB** class defines operations for converting object references to strings and converting strings to object references. The **ORB** also defines operations used by the Dynamic Invocation Interface for creating lists (NVLists) and determining the default context.

File Stem

orb

Base

SOMObject

Metaclass

SOMMSingleInstance

Ancestor Classes

SOMObject

New Methods

create_list
create_operation_list
get_default_context
object_to_string
string_to_object

create_list Method

Purpose

Creates an **NVList** of the specified size.

IDL Syntax

ORBStatus create_list (
in long *count*,
out NVList *new_list***);**

Description

Creates an **NVList** list of the specified size, typically for use in **Request**s.

Ownership of the allocated *new_list* is transferred to the caller.

Parameters

receiver	A pointer to the **ORB** object.
env	A pointer to the **Environment** structure for the method caller.
count	An integer representing the number of elements to allocate for the list.
new_list	A pointer to the address where the method will store a pointer to the allocated **NVList** object.

Return Value

The **create_list** method returns an **ORBStatus** value representing the return code of the operation.

Example

```
#include <somd.h>

Environment ev;
long nargs = 5;
NVList arglist;
ORBStatus rc;
...
rc = _create_list(SOMD_ORBObject, &ev, nargs, &arglist);
```

Original Class

ORB

Related Information

Methods: create_operation_list

create_operation_list Method

Purpose

Creates an **NVList** initialized with the argument descriptions for a given operation.

IDL Syntax

ORBStatus create_operation_list (
in OperationDef *oper*,
out NVList *new_list*);

Description

Creates an **NVList** list for the specified operation, for use in **Request**s invoking that operation.

Parameters

receiver — A pointer to the **ORB** object.

env — A pointer to the **Environment** structure for the method caller.

oper — A pointer to the **OperationDef** object representing the operation for which the **NVList** is to be initialized.

new_list — A pointer to where the method will store a pointer to the resulting argument list.

Return Value

The **create_operation_list** method returns an **ORBStatus** value representing the return code of the operation.

Ownership of the allocated *new_list* is transferred to the caller.

Example

```
#include <somd.h>

Environment ev;
OperationDef opdef;
NVList arglist;
long rc;

/* Get the OperationDef from the Interface Repository. */
opdef = _lookup_id(SOM_InterfaceRepository,
                   &ev, ”Foo:methodLong”);
/* Create a NamedValue list for the operation. */
rc= _create_operation_list(SOMD_ORBObject, &ev, opdef, &arglist);
```

Original Class

ORB

Related Information

Methods: create_list

get_default_context Method

Purpose

Returns the default process **Context** object.

IDL Syntax

ORBStatus get_default_context (
out Context *ctx***);**

Description

The **get_default_context** method gets the default process **Context** object.

Ownership of the allocated **Context** object is transferred to the caller.

Parameters

receiver	A pointer to the **ORB** object.
env	A pointer to the **Environment** structure for the method caller.
ctx	A pointer to where the method will store a pointer to the returned **Context** object.

Return Value

The **get_default_context** method returns an **ORBStatus** return code: 0 indicates success, while a non-zero value is a DSOM error code (see Appendix A of the *SOM Toolkit User's Guide*).

Example

```
#include <somd.h>

Environment ev;
Context cxt;
long rc;
...
rc = _get_default_context(SOMD_ORBObject, &ev, &cxt);
```

Original Class

ORB

object_to_string Method

Purpose

Converts an object reference to an external form (string) which can be stored outside the ORB.

IDL Syntax

string object_to_string (
in SOMDObject *obj***);**

Description

The **object_to_string** method converts the object reference to a form (string) which can be stored externally.

Ownership of allocated memory is transferred to the caller. The caller should free the result using the **ORBfree** function, rather than the **SOMFree** function.

Parameters

receiver	A pointer to the **ORB** object.
env	A pointer to the **Environment** structure for the method caller.
obj	A pointer to a **SOMDObject** object representing the reference to be converted.

Return Value

The **object_to_string** method returns a string representing the external (string) form of the referenced object.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
string objrefstr;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);

/* create a remote Car object */
car = _somdNewObject(SOMD_ObjectMgr, &ev, "Car", "");

/* save the reference to the object */
objrefstr = _object_to_string(SOMD_ORBObject, &ev, car);
FileWrite("/u/joe/mycar", objrefstr);
```

Original Class

ORB

Related Information

Methods: string_to_object

string_to_object Method

Purpose

Converts an externalized (string) form of an object reference into an object reference.

IDL Syntax

SOMDObject string_to_object (
in string *str***);**

Description

The **string_to_object** method converts the externalized (string) form of an object reference into an object reference.

Parameters

receiver	A pointer to the **ORB** object.
env	A pointer to the **Environment** structure for the method caller.
str	A pointer to a character string representing the externalized form of the object reference.

Return Value

The **string_to_object** method returns a **SOMDObject** object.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
string objrefstr;
...
/* restore proxy from its string form */
FileRead(”/u/joe/mycar”, &objrefstr);
car = _string_to_object(SOMD_ORBObject, &ev, objrefstr);
```

Original Class

ORB

Related Information

Methods: object_to_string

Principal Class

Description

The **Principal** class defines attributes which identify the user ID and host name of the originator of a specific request. This information is typically used for access control.

A **Principal** object is returned by the **get_principal** method of the SOM Object Adapter. The parameters of the **get_principal** method identify the environment and target object associated with a particular request — the **SOMOA** uses this information to create a **Principal** object which identifies the caller.

* **Note:** Details of the **Principal** object are not currently defined in the CORBA 1.1 specification; the attributes which have been defined are required by DSOM.

File Stem

principl

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

Attributes

Listed below is each available attribute, with its corresponding type in parentheses, followed by a description of its purpose:

userName (string)

Identifies the name of the user associated with the request invocation. (Currently, this value is obtained from the USER environment variable in the process which invoked the request.)

hostName (string)

Identifies the name of the host from where the request originated. (Currently, this value is obtained from the HOSTNAME environment variable in the process which invoked the request.)

Request Class

Description

The **Request** class implements the CORBA Request object described in section 6.2 on page 108 of CORBA 1.1. The **Request** object is used by the dynamic invocation interface to dynamically create and issue a request to a remote object. **Request** objects are created by the **create_request** method in **SOMDObject**.

File Stem

request

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

add_arg
destroy*
get_response
invoke
send

(* The **destroy** method was defined as **delete** in CORBA 1.1, which conflicts with the **delete** operator in C++. However, there is a **Request_delete** macro defined for CORBA compatibility.)

Overridden Methods

somInit
somUninit

add_arg Method

Purpose

Incrementally adds an argument to a **Request** object.

IDL Syntax

ORBStatus add_arg (
in Identifier *name***,**
in TypeCode *arg_type***,**
in void* *value***,**
in long *len***,**
in Flags *arg_flags***);**

Description

The **add_arg** method incrementally adds an argument to a **Request** object. The **Request** object must have been created using the **create_request** method with an empty argument list.

Parameters

receiver	A pointer to a **Request** object.
env	A pointer to the **Environment** structure for the method caller.
name	An identifier representing the name of the argument to be added.
arg_type	The typecode for the argument to be added.
value	A pointer to the argument value to be added.
len	The length of the argument.
arg_flags	A **Flags** bitmask (unsigned long). The *arg_flags* parameter may take one of the following values to indicate parameter direction:

ARG_IN	The argument is input only.
ARG_OUT	The argument is output only
ARG_INOUT	The argument is input/output

In addition, *arg_flags* may also contain the following values:

IN_COPY_VALUE	An internal copy of the argument is to be made and used.
DEPENDENT_LIST	Indicates that a specified sublist must be freed when the parent list is freed.

Return Value

The **add_arg** method returns an **ORBStatus** value representing the return code of the operation.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code builds a request to execute the call:
 *      result = methodLong(fooObj, &ev, 100,200);
 *using the DII.
 */

Environment ev;
OperationDef opdef;
Description desc;
OperationDescription *opdesc;
long rc;
long value1 = 100;
long value2 = 200;
Foo fooObj;
Request reqObj;
NamedValue result;

/* Get the OperationDef from the Interface Repository. */
opdef = _lookup_id(SOM_InterfaceRepository,
                   &ev, ”Foo::methodLong”);

/* Get the operation description structure. */
desc = _describe(opdef, &ev);
opdesc = (OperationDescription *) desc.value._value;

/* Fill in the TypeCode field for result. */
result.argument._type = opdesc->result;

/* Create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, ”methodLong”,
                     (NVList *)NULL, &result, &reqObj, (Flags)0);

/* Add arg1 info onto the request */
_add_arg(reqObj, &ev,
          ”inLong”, TC_long, &value1, sizeof(long), (Flags)0);
/* Add arg2 info onto the request */
_add_arg(reqObj, &ev,
          ”inoutLong”, TC_long, &value2, sizeof(long), (Flags)0);
```

Original Class

Request

destroy Method (for a Request object)

Purpose

Deletes the memory allocated by the ORB for a **Request** object.

IDL Syntax

ORBStatus destroy ( );

Description

The **destroy** method deletes the **Request** object and all associated memory.

NOTE: This method is called “delete” in the CORBA 1.1 specification. However, the word “delete” is a reserved operator in C++, so the name “destroy” was chosen as an alternative. For CORBA compatibility, a macro defining **Request_delete** as an alias for **destroy** has been included in the C header files.

Parameters

receiver	A pointer to a **Request** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **destroy** method returns an **ORBStatus** value representing the return code of the operation.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code sends a request to execute the call:
 *      result = methodLong(fooObj, &ev, 100,200);
 * using the DII without waiting for the result. Then, later,
 * waits for and then uses the result.
 */
Environment ev;
NVList arglist;
long rc;
Foo fooObj;
Request reqObj;
NamedValue result;

/* see the Example code for invoke to see how the request
 * is built
 */

/* Create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, "methodLong",
                     arglist, &result, &reqObj, (Flags)0);

/* Finally, send the request */
rc = _send(reqObj, &ev, (Flags)0);

/* do some work, i.e. don't wait for the result */

/* wait here for the result of the request */
rc = _get_response(reqObj, &ev, (Flags)0);

/* use the result */
if (result->argument._value == 9600) {...}

/* throw away the reqObj */
_destroy(reqObj, &ev);
```

Original Class

Request

Related Information

Methods: invoke, **send**, **get_response**

get_response Method

Purpose

Determines whether an asynchronous **Request** has completed.

IDL Syntax

```
ORBStatus  get_response (
                              in Flags response_flags);
```

Description

The **get_response** method determines whether the asynchronous **Request** has completed.

Parameters

receiver	A pointer to a **Request** object.
env	A pointer to the **Environment** structure for the method caller.
response_flags	A **Flags** bitmask (unsigned long) containing control information for the **get_response** method. The *response_flags* argument may have the following value:

RESP_NO_WAIT	Indicates the caller does not want to wait for a response.

Return Value

The **get_response** method returns an **ORBStatus** value representing the return code of the operation.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code sends a request to execute the call:
 *      result = methodLong(fooObj, &ev, 100,200);
 * using the DII without waiting for the result. Then, later,
 * waits for and then uses the result.
 */

Environment ev;
NVList arglist;
long rc;
Foo fooObj;
Request reqObj;
NamedValue result;

/* see the Example code for invoke to see how the request
 * is built
 */

/* Create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, "methodLong",
                     arglist, &result, &reqObj, (Flags)0);

/* Finally, send the request */
rc = _send(reqObj, &ev, (Flags)0);

/* do some work, i.e. don't wait for the result */

/* wait here for the result of the request */
rc = _get_response(reqObj, &ev, (Flags)0);

/* use the result */
if (result->argument._value == 9600) {...}
```

Original Class

Request

Related Information

Methods: invoke, **send**

Macros: Request_delete

invoke Method

Purpose

Invokes a **Request** synchronously, waiting for the response.

IDL Syntax

ORBStatus invoke (
in Flags *invoke_flags***);**

Description

The **invoke** method sends a **Request** synchronously, waiting for the response.

Parameters

receiver A pointer to a **Request** object.

env A pointer to the **Environment** structure for the method caller.

invoke_flags A **Flags** bitmask (unsigned long) representing control information for the **invoke** method. There are currently no flags defined for the **invoke** method.

Return Value

The **invoke** method returns an **ORBStatus** value representing the return code of the operation.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code builds and then invokes
 * a request to execute the call:
 *      result = methodLong(fooObj, &ev, 100,200);
 * using the DII.
 */

Environment ev;
OperationDef opdef;
Description desc;
OperationDescription *opdesc;
NVList arglist;
long rc;
long value1 = 100;
long value2 = 200;
Foo fooObj;
Request reqObj;
NamedValue result;
Identifier name;
TypeCode tc;
void *dummy;
long dummylen;
Flags flags;
```

```
/* Get the OperationDef from the Interface Repository. */
opdef = _lookup_id(SOM_InterfaceRepository,
                   &ev, "Foo::methodLong");
/* Create a NamedValue list for the operation. */
rc= _create_operation_list(SOMD_ORBObject, &ev, opdef, &arglist);

/* Insert arg1 info into arglist */
_get_item(arglist, &ev,
          0, &name, &tc, &dummy, &dummylen, &flags);
_set_item(arglist,&ev,0, name, tc, &value1, sizeof(long), flags);

/* Insert arg2 info into arglist */
_get_item(arglist, &ev,
          1, &name, &tc, &dummy, &dummylen, &flags);
_set_item(arglist,&ev,1, name, tc, &value2, sizeof(long), flags);

/* Get the operation description structure. */
desc = _describe(opdef, &ev);
opdesc = (OperationDescription *) desc.value._value;

/* Fill in the TypeCode field for result. */
result.argument._type = opdesc->result;

/* Create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, "methodLong",
                     arglist, &result, &reqObj, (Flags)0);

/* Finally, invoke the request */
rc = _invoke(reqObj, &ev, (Flags)0);

/* Print results */
printf("result: %d, value2: %d\n",
       *(long*)(result.argument._value),
       value2);
```

Original Class

Request

Related Information

Methods: send, **get_response**

Macros: Request_delete

send Method

Purpose

Invokes a **Request** asynchronously.

IDL Syntax

ORBStatus send (
in Flags *invoke_flags***);**

Description

The **send** method invokes the **Request** asynchronously. The response must eventually be checked by invoking either the **get_response** method or the **get_next_response** function.

Parameters

receiver	A pointer to a **Request** object.
env	A pointer to the **Environment** structure for the method caller.
invoke_flags	A **Flags** bitmask (unsigned long) containing **send** method control information. The argument *invoke_flags* can have the following value.

INV_NO_RESPONSE	Indicates that the invoker does not intend to wait for a response, nor does it expect any of the output arguments (**inout** or **out**) to be updated.

Return Value

The **send** method returns an **ORBStatus** value representing the return code from the operation.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code sends
 * a request to execute the call:
 *      result = methodLong(fooObj, &ev, 100,200);
 * using the DII.
 */

Environment ev;
NVList arglist;
long rc;
Foo fooObj;
Request reqObj;
NamedValue result;

/* see the Example code for invoke to see how the request
 * is built
 */

/* Create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, "methodLong",
                     arglist, &result, &reqObj, (Flags)0);

/* Finally, send the request */
rc = _send(reqObj, &ev, (Flags)0);
```

Original Class

Request

Related Information

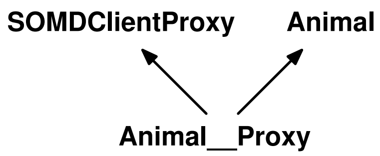
Methods: invoke, **get_response**

Macros: Request_delete

SOMDClientProxy Class

Description

The **SOMDClientProxy** class implements DSOM proxy objects in Clients. **SOMDClientProxy** overrides the usual **somDispatch** methods with versions that build a DSOM **Request** for remote invocation and dispatch it to the remote object. It is intended that the implementation of this "generic" proxy class will be used to derive specific proxy classes via multiple inheritance. The remote dispatch method is inherited from this client proxy class, while the desired interface — and language bindings — are inherited from the target class (but not the implementation).



File Stem

somdcprx

Base

SOMDObject

Metaclass

SOMClass

Ancestor Classes

SOMObject, SOMDObject

New Methods

somdProxyFree*
somdProxyGetClass*
somdProxyGetClassName*
somdTargetFree*
somdTargetGetClass*
somdTargetGetClassName*
somdReleaseResources*

(* This class and its methods were added by DSOM to supplement the published CORBA 1.1 interfaces.)

Overridden methods

create_request
create_request_args
is_proxy
release
somDispatch
somDispatchA, somDispatchD, somDispatchL, somDispatchV
somFree
somGetClass
somGetClassName
somInit
somUninit

somdProxyFree Method

Purpose

Executes **somFree** on the local proxy object.

IDL Syntax

void somdProxyFree ( );

Description

The **somdProxyFree** method executes the **somFree** method call on the local proxy object. This method has been provided when the application program wants to be explicit about freeing the proxy object vs. the target object.

Parameters

receiver	A pointer to the **SOMDClientProxy** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

somdProxyFree has no return value.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
string somdObjectId;
...
/* restore proxy from its string form */
FileRead("/u/joe/mycar", &somdObjectId);
car = _somdGetObjectFromId(SOMD_ObjectMgr, &ev, somdObjectId);
...
_somdProxyFree(car, &ev);
```

Original Class

SOMDClientProxy

Related Information

Methods: release, somdReleaseObject

somdProxyGetClass Method

Purpose

Returns the class object for the local proxy object.

IDL Syntax

SOMClass somdProxyGetClass ( );

Description

The **somdProxyGetClass** method executes the **somGetClass** method call on the local proxy object and returns a pointer to the proxy's class object. This method has been provided when the application program wants to be explicit about getting the class object for the proxy object vs. the target object.

Parameters

receiver	A pointer to the **SOMDClientProxy** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **somdProxyGetClass** method returns a pointer to the class object for the local proxy object.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
SOMClass carProxyClass;
string somdObjectId;
...
/* restore proxy from its string form */
FileRead(”/u/joe/mycar”, &somdObjectId);
car = _somdGetObjectFromId(SOMD_ObjectMgr, &ev, somdObjectId);
...
carProxyClass = _somdProxyGetClass(car, &ev);
```

Original Class

SOMDClientProxy

somdProxyGetClassName Method

Purpose

Returns the class name for the local proxy object.

IDL Syntax

string somdProxyGetClassName ( );

Description

The **somdProxyGetClassName** method executes the **somGetClassName** method call on the local proxy object and returns the proxy's class name. This method has been provided when the application program wants to be explicit about getting the class name of the proxy object vs. the target object.

Parameters

receiver	A pointer to the **SOMDClientProxy** object for the desired remote target object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **somdProxyGetClassName** method returns a string containing the class name of the local proxy object.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
string carProxyClassName;
string somdObjectId;
...
/* restore proxy from its string form */
FileRead(”/u/joe/mycar”, &somdObjectId);
car = _somdGetObjectFromId(SOMD_ObjectMgr, &ev, somdObjectId);
...
carProxyClassName = _somdProxyGetClassName(car, &ev);
```

Original Class

SOMDClientProxy

somdReleaseResources Method

Purpose

Instructs a proxy object to release any memory it is holding as a result of a remote method invocation in which a parameter or result was designated as "object-owned".

IDL Syntax

void somdReleaseResources ();

Description

The **somdReleaseResources** method instructs a proxy object to release any memory it is holding as a result of a remote method invocation in which a parameter or result was designated as "object-owned".

When a DSOM client program makes a remote method invocation, via a proxy, and the method being invoked has an object-owned parameter or return result, the client-side memory associated with the parameter/result will be owned by the caller's proxy, and the server-side memory will be owned by the remote object. The memory owned by the caller's proxy will be freed when the proxy is released by the client program. (The time at which the server-side memory will be freed depends on the implementation of the remote object.)

A DSOM client can also instruct a proxy object to free all memory that it owns on behalf of the client without releasing the proxy (assuming that the client program is finished using the object-owned memory), by invoking the **somdReleaseResources** method on the proxy object. Calling **somdReleaseResources** can prevent unused memory from accumulating in a proxy.

For example, consider a client program repeatedly invoking a remote method "get_string", which returns a string that is designated (in SOM IDL) as "object-owned". The proxy on which the method is invoked will store the memory associated with all of the returned strings, even if the strings are not unique, until the proxy is released. If the client program only uses the last result returned from "get_string", then unused memory accumulates in the proxy. The client program can prevent this by invoking **somdReleaseResources** on the proxy object periodically (for example, each time it finishes using the result of the last "get_string" call).

Parameters

receiver A pointer to the **SOMDClientProxy** object to release resources.

ev A pointer to the **Environment** structure for the method call.

Return Value

None.

Example

```
string mystring;
...
/* remote invocation of get_string on proxy x,
 * where method get_string has the SOM IDL modifier
 * "object_owns_result".
 */
mystring = X_get_string(x, ev);

/* ... use mystring ... */
```

```
/* when finished using mystring, instruct the
 * proxy that it can free it.
 */
_somdReleaseResources(x, ev);
```

Original Class

SOMDClientProxy

Related Information

Methods: release

somdTargetFree Method

Purpose

Forwards the **somFree** method call to the remote target object.

IDL Syntax

void somdTargetFree ( );

Description

The **somdTargetFree** method forwards the **somFree** method call to the remote target object. This method has been provided when the application program wants to be explicit about freeing the remote target object vs. the proxy object.

Parameters

receiver	A pointer to the **SOMDClientProxy** object for the desired remote target object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

somdTargetFree has no return value.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
string somdObjectId;
...
/* restore proxy from its string form */
FileRead("/u/joe/mycar", &somdObjectId);
car = _somdGetObjectFromId(SOMD_ObjectMgr, &ev, somdObjectId);
...
_somdTargetFree(car, &ev);
```

Original Class

SOMDClientProxy

Related Information

Methods: release, somdDestroyObject

somdTargetGetClass Method

Purpose

Returns (a proxy for) the class object for the remote target object.

IDL Syntax

SOMClass somdTargetGetClass ( );

Description

The **somdTargetGetClass** method forwards the **somGetClass** method call to the remote target object and returns a pointer to the class object for that object. This method has been provided when the application program wants to be explicit about getting the class object for the remote target object vs. the local proxy.

Parameters

receiver	A pointer to the **SOMDClientProxy** object for the desired remote target object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **somdTargetGetClass** method returns a pointer to the class object for the remote target object.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
SOMClass carClass;
string somdObjectId;
...
/* restore proxy from its string form */
FileRead(”/u/joe/mycar”, &somdObjectId);
car = _somdGetObjectFromId(SOMD_ObjectMgr, &ev, somdObjectId);
...
carClass = _somdTargetGetClass(car, &ev);
```

Original Class

SOMDClientProxy

Related Information

Methods: somdProxyGetClass

somdTargetGetClassName Method

Purpose

Returns the class name for the remote target object.

IDL Syntax

string somdTargetGetClassName ( );

Description

The **somdTargetGetClassName** method forwards the **somGetClassName** method call to the remote target object and returns the class name for that object. This method has been provided when the application program wants to be explicit about getting the class name of the remote target object vs. the proxy object.

Parameters

receiver	A pointer to the **SOMDClientProxy** object for the desired remote target object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **somdTargetGetClassName** method returns a string containing the class name of the remote target object.

Example

```
#include <somd.h>
#include <car.h>

Environment ev;
Car car;
string carClassName;
string somdObjectId;
...
/* restore proxy from its string form */
FileRead("/u/joe/mycar", &somdObjectId);
car = _somdGetObjectFromId(SOMD_ObjectMgr, &ev, somdObjectId);
...
carClassName = _somdTargetGetClassName(car, &ev);
```

Original Class

SOMDClientProxy

Related Information

Methods: somdProxyGetClassName

SOMDObject Class

Description

The **SOMDObject** class implements the methods that can be applied to all CORBA object references: for example, **get_implementation**, **get_interface**, **is_nil, duplicate**, and **release**. (In the CORBA 1.1 specification, these methods are described in Chapter 8.)

In DSOM, there is also another derivation of this class: **SOMDClientProxy**. This subclass inherits the implementation of **SOMDObject**, but extends it by overriding **somDispatch** with a "remote dispatch" method, and caches the binding to the server process. Whenever a remote object is accessed, it is represented in the client process by a **SOMDClientProxy** object.

File Stem

somdobj

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

create_request
create_request_args*
duplicate
get_implementation
get_interface
is_constant*
is_nil
is_proxy*
is_SOM_ref*
release

(* These methods were added by DSOM to supplement
the published CORBA 1.1 interfaces.)

Overridden methods

somInit
somUninit
somDumpSelfInt

create_request Method

Purpose

Creates a request to execute a particular operation on the referenced object.

IDL Syntax

ORBStatus create_request (
in Context *ctx***,**
in Identifier *operation***,**
in NVList *arg_list***,**
inout NamedValue *result***,**
out Request *request***,**
in Flags *req_flags***);**

Description

The **create_request** method creates a request to execute a particular operation on the referenced object. [For more information on the **create_request** call, see CORBA 1.1 page 109.]

In DSOM, this method is meaningful only when invoked on a **SOMDClientProxy** object. If invoked on a **SOMDObject** which is not a client proxy, an exception is returned.

Parameters

receiver	A pointer to a **SOMDObject** object.
env	A pointer to the **Environment** structure for the method caller.
ctx	A pointer to the **Context** object of the requested operation.
operation	The name of the operation to be performed on the target object, *receiver*.
arg_list	A pointer to a list of arguments (**NVList**). If this argument is NULL, the argument list can be assembled by repeated calls to the **add_arg** method on the **Request** object created by calling this method.
result	A pointer to a **NamedValue** structure where the result of applying *operation* to *receiver* should be stored.
request	A pointer to storage for the address of the created **Request** object.
req_flags	A **Flags** bitmask (unsigned long) that may contain the following flag value:

OUT_LIST_MEMORY	Indicates that any out-arg memory is associated with the argument list. When the list structure is freed, any associated out-arg memory is also freed. If OUT_LIST_MEMORY is specified, an argument list must also have been specified on the **create_request** call.

Return Value

The **create_request** method returns an **ORBStatus** value as the status code for the request.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code builds a request to execute the call:
 *      result = methodLong(fooObj, &ev, 100,200);
 *using the DII.
 */

Environment ev;
OperationDef opdef;
Description desc;
OperationDescription *opdesc;
NVList arglist;
long rc;
long value1 = 100;
long value2 = 200;
Foo fooObj;
Request reqObj;
NamedValue result;
Identifier name;
TypeCode tc;
void *dummy;
long dummylen;
Flags flags;

/* Get the OperationDef from the Interface Repository. */
opdef = _lookup_id(SOM_InterfaceRepository,
                   &ev, "Foo::methodLong");
/* Create a NamedValue list for the operation. */
rc= _create_operation_list(SOMD_ORBObject, &ev, opdef, &arglist);

/* Insert arg1 info into arglist */
_get_item(arglist, &ev,
          0, &name, &tc, &dummy, &dummylen, &flags);
_set_item(arglist,&ev,0, name, tc, &value1, sizeof(long), flags);

/* Insert arg2 info into arglist */
_get_item(arglist, &ev,
          1, &name, &tc, &dummy, &dummylen, &flags);
_set_item(arglist,&ev,1, name, tc, &value2, sizeof(long), flags);

/* Get the operation description structure. */
desc = _describe(opdef, &ev);
opdesc = (OperationDescription *) desc.value._value;

/* Fill in the TypeCode field for result. */
result.argument._type = opdesc->result;

/* Finally, create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, "methodLong",
                     arglist, &result, &reqObj, (Flags)0);
```

Original Class

SOMDObject

Related Information

Methods: create_request_args, create_list, create_operation_list

create_request_args Method

Purpose

Creates an argument list appropriate for the specified operation.

IDL Syntax

ORBStatus create_request_args (
in Identifier *operation***,**
out NVList *arg_list.*
out NamedValue *result***);**

Description

The **create_request_args** method creates the appropriate *arg_list* (**NVList**) for the specified operation. It is similar in function to the **create_operation_list** method. Its value is that it also creates the result structure whereas **create_operation_list** does not.

In DSOM, this method is meaningful only when invoked on a **SOMDClientProxy** object. If invoked on a **SOMDObject** which is not a client proxy, an exception is returned.

Parameters

receiver	A pointer to the **SOMDObject** object to create the request.
env	A pointer to the **Environment** structure for the method caller.
operation	The Identifier of the operation for which the argument list is being created.
arg_list	A pointer to the location where the method will store a pointer to the resulting argument list.
result	A pointer to the **NamedValue** structure which will be used to hold the result. The *result*'s type field is filled in with the **TypeCode** of the expected result.

Return Value

The **create_request_args** method returns an **ORBStatus** value representing the return code of the request.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>
#include <foo.h>   /* provided by user */

/* assume following method declaration in interface Foo:
 *      long methodLong  (in long inLong,inout long inoutLong);
 * then the following code builds a request to execute the call:
 *      result = methodLong(fooObj, &ev, 100,200);
 * using the DII.
 */

Environment ev;
OperationDef opdef;
Description desc;
OperationDescription *opdesc;
NVList arglist;
long rc;
long value1 = 100;
long value2 = 200;
Foo fooObj;
Request reqObj;
NamedValue result;
Identifier name;
TypeCode tc;
void *dummy;
long dummylen;
Flags flags;

/* Get the OperationDef from the Interface Repository. */
opdef = _lookup_id(SOM_InterfaceRepository,
                   &ev, "Foo::methodLong");
/* Create a NamedValue list for the operation. */
rc= _create_request_args(fooObj, &ev,
                         "methodLong", &arglist, &result);

/* Insert arg1 info into arglist */
_get_item(arglist, &ev,
          0, &name, &tc, &dummy, &dummylen, &flags);
_set_item(arglist,&ev,0, name, tc, &value1, sizeof(long), flags);

/* Insert arg2 info into arglist */
_get_item(arglist, &ev,
          1, &name, &tc, &dummy, &dummylen, &flags);
_set_item(arglist,&ev,1, name, tc, &value2, sizeof(long), flags);

/* Finally, create the Request, reqObj */
rc = _create_request(fooObj, &ev, (Context *)NULL, "methodLong",
                     arglist, &result, &reqObj, (Flags)0);
```

Original Class

SOMDObject

Related Information

Methods: duplicate, **release**, **create_request, create_operation_list**

duplicate Method

Purpose

Makes a duplicate of an object reference.

IDL Syntax

SOMDObject duplicate ( );

Description

The **duplicate** method makes a duplicate of the object reference. The **release** method should be called to free the object.

Parameters

receiver	A pointer to a **SOMDObject** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **duplicate** method returns a **SOMDObject** that is a duplicate of the *receiver*. *Ownership* of the returned object is transferred to the caller.

Example

```
#include <somd.h>

Environment ev;
SOMObject obj;
SOMDObject objref1, objref2;
...
objref1 = _create_SOM_ref(SOMD_SOMOAObject, &ev, obj);
objref2 = _duplicate(objref1,&ev);
...
_release(objref2,&ev);
```

Original Class

SOMDObject

Related Information

Methods: release, **create, create_constant, create_SOM_ref**

get_implementation Method

Purpose

Returns the implementation definition for the referenced object.

IDL Syntax

ImplementationDef get_implementation ( );

Description

The **get_implementation** method returns the implementation definition object for the referenced object.

Parameters

receiver	A pointer to a **SOMDObject** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **get_implementation** method returns the **ImplementationDef** object for the *receiver*. *Ownership* of the returned object is transferred to the caller.

Example

```
#include <somd.h>

long flags;
Environment ev;
SOMDObject objref;
ImplementationDef impldef;
...
impldef = _get_implementation(objref,&ev);
flags = __get_impl_flags(impldef,&ev);
```

Original Class

SOMDObject

Related Information

Methods: get_interface

get_interface Method

Purpose

Returns the interface definition object for the referenced object.

IDL Syntax

InterfaceDef get_interface ( );

Description

The **get_interface** method returns the interface definition object for the referenced object.

Parameters

receiver	A pointer to a **SOMDObject** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **get_interface** method returns a pointer to the **InterfaceDef** object associated with the reference *receiver*. *Ownership* of the **InterfaceDef** object is passed to the caller.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>

Environment ev;
SOMDObject objref;
InterfaceDef intf;
...
intf = _get_interface(objref,&ev);
```

Original Class

SOMDObject

Related Information

Methods: get_implementation

is_constant Method

Purpose

Tests to see if the object reference is a constant (that is, its **ReferenceData** is a constant value associated with the reference).

IDL Syntax

boolean is_constant ( );

Description

The **is_constant** method tests to see if the object reference was created using the **create_constant** method in the **SOMOA** class.

Parameters

receiver	A pointer to a **SOMDObject** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **is_constant** method returns TRUE if the object reference was generated by the method **create_constant.** Otherwise, **is_constant** returns FALSE.

Example

```
#include <somd.h>

Environment ev;
SOMDObject objref;
...

/* This code might be part of the code
 * that overrides the somdSOMObjFromRef method, i.e.
 * in an implementation of a subclass of SOMDServer called
 * myServer
 */

if (_is_constant(objref, &ev))
    id = _get_id(objref, &ev);

...
```

Related Information

Methods: create, create_constant, is_proxy, is_SOM_ref, is_nil

is_nil Method

Purpose

Tests to see if the object reference is nil.

IDL Syntax

boolean is_nil ( );

Description

The **is_nil** method tests to see if the specified object reference is nil.

Parameters

receiver A pointer to any object, either a **SOMObject** or a **SOMDObject**. The pointer can be NULL.

env A pointer to the **Environment** structure for the method caller.

Return Value

The **is_nil** method returns TRUE if the object reference is empty. Otherwise, **is_nil** returns FALSE.

Example

```
#include <somd.h>
Environment ev;
SOMDObject objref;
SOMObject somobj;
...
/* This code might be part of the code
 * that overrides the somdSOMObjFromRef method, i.e.
 * in an implementation of a subclass of SOMDServer called
 * myServer
 */
if (_is_nil(objref, &ev) ||
     _somIsA(objref, SOMDClientProxyNewClass(0, 0)) ||
     _is_SOM_ref(objref, &ev)) {
  somobj = myServer_parent_SOMDServer_somdSOMObjFromRef
             (somSelf, &ev, objref);
}
else {
  /* do the myServer-specific stuff to create/find somobj here */
}
return somobj;
```

Related Information

Methods: create, is_constant, is_proxy, is_SOM_ref

is_proxy Method

Purpose

Tests to see if the object reference is a proxy.

IDL Syntax

boolean is_proxy( );

Description

The **is_proxy** method tests to see if the specified object reference is a proxy object.

Parameters

receiver	A pointer to a **SOMDObject** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

The **is_proxy** method returns TRUE if the object reference is a proxy object. Otherwise, **is_proxy** returns FALSE.

Example

```
#include <somd.h>

SOMDObject objref;
Environment ev;
Context ctx;
NVlist arglist;
NamedValue result;
Request reqObj;
...
if (_is_proxy(objref, &ev)) {
    /* create a remote request for target object */
...
   rc = _create_request(obj, &ev, ctx,
                         "testMethod", arglist, &result, &reqObj,
                         (Flags)0);
}
...
```

Original Class

SOMDObject

Related Information

Methods: is_nil, is_constant, is_SOM_ref, string_to_object

is_SOM_ref Method

Purpose

Tests to see if the object reference is a simple reference to a SOM object.

IDL Syntax

boolean is_SOM_ref ( );

Description

The **is_SOM_ref** method tests to see if the specified object reference is a simple (transient) reference to a SOM object.

Parameters

receiver A pointer to a **SOMDObject** object.

env A pointer to the **Environment** structure for the method caller.

Return Value

The **is_SOM_ref** method returns TRUE if the object reference is a simple (transient) reference to a SOM object. Otherwise, **is_SOM_ref** returns FALSE.

Example

```
#include <somd.h>

SOMDObject objref;
Environment ev;
SOMObject obj;
...
if (_is_SOM_ref(objref, &ev))
    /* we know objref is a simple reference, so we can ... */
    obj = _get_SOM_object(SOMD_SOMOAObject, &ev, objref);
...
```

Original Class

SOMDObject

Related Information

Methods: create_SOM_ref, get_SOM_object, is_proxy, is_constant, is_nil

release Method

Purpose

Releases the memory associated with the specified object reference.

IDL Syntax

void release ( );

Description

The **release** method releases the memory associated with the object reference.

Parameters

receiver	A pointer to a **SOMDObject** object.
env	A pointer to the **Environment** structure for the method caller.

Return Value

None.

Example

```
#include <somd.h>

SOMDObject objref;
Environment ev;
SOMObject obj;
...
objref = _create_SOM_ref(SOMD_SOMOAObject, &ev, obj);
...
_release(objref, &ev);
```

Original Class

SOMDObject

Related Information

Methods: duplicate, **somdReleaseObject, somdProxyFree, create, create_constant, create_SOM_ref, somdReleaseResources**

SOMDObjectMgr Class

Description

The **SOMDObjectMgr** class is derived from **ObjectMgr** class and provides the DSOM implementations for the **ObjectMgr** methods.

File Stem

somdom

Base

ObjectMgr

Metaclass

SOMMSingleInstance

Ancestor Classes

ObjectMgr, SOMObject

Attribute

Listed below is an available **SOMDObjectMgr** attribute, with its corresponding type in parentheses, followed by a description of its purpose:

somd21somFree (boolean)

Determines whether or not **somFree**, when invoked on a proxy object, will free the proxy object along with the remote object. The default value is FALSE, indicating that only the remote object will be freed when **somFree** is invoked on a proxy object. Setting this attribute to TRUE as part of client-program initialization, for example,

```
__set_somd21somdFree(SOMD_ObjectMgr, ev, TRUE);
```

has the effect that all subsequent invocations of **somFree** on proxy objects will free both the remote object and the proxy.

New Methods

somdFindAnyServerByClass*
somdFindServer*
somdFindServersByClass*
somdFindServerByName*

(* This class and its methods were added by DSOM to supplement the published CORBA 1.1 interfaces.)

Overridden Methods

somdDestroyObject
somdGetIdFromObject
somdGetObjectFromId
somdNewObject
somdReleaseObject
somInit

somdFindAnyServerByClass Method

Purpose

Finds a server capable of creating the specified object.

IDL Syntax

SOMDServer somdFindAnyServerByClass (
in Identifier *objclass***);**

Description

The **somdFindAnyServerByClass** method finds a server capable of creating an object of the specified type with the specified properties.

Parameters

receiver	A pointer to a **SOMDObjectMgr** object.
env	A pointer to the **Environment** structure for the method caller.
objclass	An **Identifier** specifying the class of the object the server needs to be able to create.

Return Value

The **somdFindAnyServerByClass** method returns a pointer to a **SOMDServer** proxy. Or, if no server can be found in the Implementation Repository that implements the specified class, NULL is returned.

Example

```
#include <somd.h>
#include <stack.h>   /* provided by user */

Stack stk;
Environment ev;
SOMDServer server;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
server =
  _somdFindAnyServerByClass(SOMD_ObjectMgr, &ev, "Stack");
stk = _somdCreateObj(server, &ev, "Stack", "");
...
_somdDestroyObject(SOMD_ObjectMgr, &ev, stk);
```

Original Class

SOMDObjectMgr

Related Information

Methods: somdFindServersByClass, somdFindServer, somdFindServerByName

somdFindServer Method

Purpose

Finds a server given its **ImplementationDef** ID.

IDL Syntax

SOMDServer somdFindServer (
in ImplId *serverid***);**

Description

The **somdFindServer** method finds a server capable of creating an object of the specified type with the specified properties.

Parameters

receiver	A pointer to a **SOMDObjectMgr** object.
env	A pointer to the **Environment** structure for the method caller.
serverid	An **ImplId** string which identifies the **ImplementationDef** of the desired server.

Return Value

The **somdFindServer** method returns a pointer to a **SOMDServer** proxy.

Example

```
#include <somd.h>
#include <stack.h>   /* provided by user */

Stack stk;
Environment ev;
SOMDServer server;
ImplId implid;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
server = _somdFindServer(SOMD_ObjectMgr, &ev, implid);
stk = _somdCreateObj(server, &ev, ”Stack”, ””);
...
_somdDestroyObject(SOMD_ObjectMgr, &ev, stk);
```

Original Class

SOMDObjectMgr

Related Information

Methods: somdFindServerByName, somdFindServersByClass, somdFindAnyServerByClass

somdFindServerByName Method

Purpose

Finds a server given its **ImplementationDef** name (alias).

IDL Syntax

SOMDServer somdFindServerByName (
in string *servername***);**

Description

The **somdFindServerByName** method finds a server with the specified name.

Parameters

receiver	A pointer to a **SOMDObjectMgr** object.
env	A pointer to the **Environment** structure for the method caller.
servername	A **string** that specifies the name of the **ImplementationDef** of the desired server.

Return Value

The **somdFindServerByName** method returns a pointer to a **SOMDServer** proxy.

Example

```
#include <somd.h>
#include <stack.h>   /* provided by user */

Stack stk;
Environment ev;
SOMDServer server;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
server =
       _somdFindServerByName(SOMD_ObjectMgr, &ev, ”stackServer”);
stk = _somdCreateObj(server, &ev, ”Stack”, ””);
...
_somdDestroyObject(SOMD_ObjectMgr, &ev, stk);
```

Original Class

SOMDObjectMgr

Related Information

Methods: somdFindServer, somdFindServersByClass, somdFindAnyServerByClass

somdFindServersByClass Method

Purpose

Finds all servers capable of creating a particular object.

IDL Syntax

sequence<SOMDServer> somdFindServersByClass (
in Identifier *objclass***);**

Description

The **somdFindServersByClass** method finds all servers capable of creating a particular object with the specified properties.

Parameters

receiver A pointer to a **SOMDObjectMgr** object.

env A pointer to the **Environment** structure for the method caller.

objclass An **Identifier** representing the type of the object the server needs to be able to create.

Return Value

The **somdFindServersByClass** method returns a sequence of **SOMDServer** objects capable of creating the specified object.

Example

```
#include <somd.h>
#include <stack.h>   /* provided by user */

Stack stk;
Environment ev;
sequence(SOMDServer) servers;
SOMDServer server;
SOMDServer chooseServer(sequence(SOMDServer) servers);

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
servers = _somdFindServersByClass(SOMD_ObjectMgr, &ev, "Stack");
server = chooseServer(servers);
stk = _somdCreateObj(server, &ev, "Stack", "");
...
_somdDestroyObject(SOMD_ObjectMgr, &ev, stk);
```

Original Class

SOMDObjectMgr

Related Information

Methods: somdFindServer, somdFindServerByName, somdFindAnyServerByClass

SOMDServer Class

Description

The **SOMDServer** class is a base class that defines and implements methods for managing objects in a DSOM server process. This includes methods for the creation and deletion of SOM objects, and for getting the SOM class object for a specified class. The **SOMDServer** class also defines and implements methods for the mapping between object references (**SOMDObject**s) and SOM objects, and dispatching methods on objects.

Application-specific methods for managing application objects can be introduced in subclasses of **SOMDServer.**

File Stem

somdserv

Base

SOMObject

Metaclass

SOMMSingleInstance

Ancestor Classes

SOMObject

New Methods

somdCreateObj*
somdDeleteObj*
somdDispatchMethod*
somdGetClassObj*
somdObjReferencesCached*
somdRefFromSOMObj*
somdSOMObjFromRef*

(***** This class and its methods were added by DSOM to supplement the published CORBA 1.1 interfaces.)

somdCreateObj Method

Purpose

Creates an object of the specified class.

IDL Syntax

SOMObject somdCreateObj (
in Identifier *objclass,*
in string *hints***);**

Description

The **somdCreateObj** method creates an object of the specified class.

Parameters

receiver	A pointer to a **SOMDServer** object capable of creating an instance of the specified class.
env	A pointer to the **Environment** structure for the method caller.
objclass	The class of the object for which an instance is to be created.
hints	A **string** which may optionally be used to specify special creation options.

Return Value

The **somdCreateObj** method returns a **SOMObject** of the class specified by *objclass*.

Example

```
#include <somd.h>
#include <stack.h>   /* provided by user */

Stack stk;
Environment ev;
SOMDServer server;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
server =
  _somdFindServerByName(SOMD_ObjectMgr, &ev,"stackServer");
stk = _somdCreateObj(server, &ev, "Stack", "");
...
_somdDestroyObject(SOMD_ObjectMgr, &ev, stk);
```

Original Class

SOMDServer

somdDeleteObj Method

Purpose

Deletes the specified object.

IDL Syntax

void somdDeleteObj (
in SOMObject *somobj* **);**

Description

The **somdDeleteObj** method deletes the specified object.

Parameters

receiver	A pointer to a **SOMDServer** object.
env	A pointer to the **Environment** structure for the method caller.
somobj	An object “managed” by the server object.

Return Value

None.

Example

```
#include <somd.h>
#include <stack.h>   /* provided by user */

Stack stk;
Environment ev;
SOMDServer server;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
server =
  _somdFindServerByName(SOMD_ObjectMgr, &ev,"stackServer");
stk = _somdCreateObj(server, &ev, "Stack", "");
...
_somdDeleteObj(server, &ev, stk);
```

Original Class

SOMDServer

somdDispatchMethod Method

Purpose

Dispatch a method on the specified SOM object.

IDL Syntax

void somdDispatchMethod (
in SOMObject *somobj,*
out somToken *retValue,*
in somId *methodId,*
in va_list *ap* **);**

Description

The **somdDispatchMethod** method is used to intercept method calls on objects in a server. When a request arrives, the request parameters are extracted from the message, and the target object is resolved. Then, the **SOMOA** dispatches the method call on the target object using the **somdDispatchMethod** method.

The default implementation will call **somDispatch** on the target object with the parameters as specified. This method can be overridden to intercept and process the method calls before they are dispatched.

Parameters

receiver	A pointer to a **SOMDServer** object.
env	A pointer to the **Environment** structure for the method caller.
somobj	A pointer to an object “managed” by the server object.
retValue	A pointer to the storage area allocated to hold the method result value, if any.
methodId	A **somId** for the name of the method which is to be dispatched.
ap	A pointer to a **va_list** array of arguments to the method call.

Return Value

The **somdDispatchMethod** method will return a result, if any, in the storage whose address is in *retValue.*

Example

```
#include <somd.h>

/* overridden somdDispatchMethod */
void somdDispatchMethod(SOMDServer *somself, Environment *ev,
                        SOMObject *somobj, somToken *retValue,
                        somId methodId, va_list ap)
{
   printf(”dispatching %s on %x\n”, SOM_StringFromId(methodId),
          somobj);
   SOMObject_somDispatch(somobj, ev, retValue, methodId, ap);
}
```

Original Class

SOMDServer

somdGetClassObj Method

Purpose

Creates a class object for the specified class.

IDL Syntax

SOMClass somdGetClassObj (
in Identifier *objclass***);**

Description

The **somdGetClassObj** method creates a class object of the specified type.

Parameters

receiver	A pointer to a **SOMDServer** object.
env	A pointer to the **Environment** structure for the method caller.
objclass	An identifier specifying the type of the class object to be created.

Return Value

The **somdGetClassObj** method returns a **SOMClass** object of the type specified.

Example

```
#include <somd.h>
#include <stack.h>   /* provided by user */

SOMClass stkclass;
Environment ev;
SOMDServer server;

SOM_InitEnvironment(&ev);
SOMD_Init(&ev);
StackNewClass(0,0);
server =
  _somdFindServerByName(SOMD_ObjectMgr, &ev,"stackServer");
stkclass = _somdGetClassObj(server, &ev, "Stack", "");
```

Original Class

SOMDServer

somdObjReferencesCached Method

Purpose

Indicates whether a server object retains ownership of the object references it creates via the **somdRefFromSOMObj** method.

IDL Syntax

boolean somdObjReferencesCached ( );

Description

The **somdObjReferencesCached** method indicates whether a server object retains ownership of the object references it creates via the **somdRefFromSOMObj** method. The default implementation returns FALSE, meaning that the server turns over ownership of the object references it creates to the caller. Subclasses of **SOMDServer** that implement object reference caching should override this method to return TRUE.

Parameters

receiver	A pointer to an object of class **SOMDServer.**
ev	A pointer to the **Environment** structure for the calling method.

Return Value

The method returns FALSE by default; overriding implementations may return TRUE to indicate that a subclass of **SOMDServer** implements object reference caching.

Example

```
SOMDobject objref;
objref = _somdRefFromSOMObj(serverObj, ev, myobj);
...
/* code to use objref */
...
if (!_somdObjReferencesCached(serverObj, ev))
    _release(objref, ev);
```

Original Class

SOMDServer

Related Information

Methods: somdRefFromSOMObj

somdRefFromSOMObj Method

Purpose

Returns an object reference corresponding to the specified SOM object.

IDL Syntax

SOMDObject somdRefFromSOMObj (
in SOMObject *somobj***);**

Description

The **somdRefFromSOMObj** method creates a simple (transient) reference to a SOM object. This method is called by **SOMOA** as part of converting the results of a local method call into a result message for a remote client.

By default the **somdRefFromSOMObj** method turns over ownership of the object reference it creates to the caller. However, if a subclass of **SOMDServer** overrides **somdRefFromSOMObj** to implement object reference caching, then that subclass should also override the method **somdObjReferencesCached** to report that caching by returning TRUE.

Parameters

receiver	A pointer to a **SOMDServer** object.
env	A pointer to the **Environment** structure for the method caller.
somobj	A pointer to the SOM object for which a DSOM reference is to be created.

Return Value

The **somdRefFromSOMObj** method returns a DSOM reference (that is, a **SOMDObject**) for the SOM object specified.

Example

```
#include <somd.h>
#include <stack.ih> /* user-generated */

SOMDObject objref;
Environment ev;
SOMObject obj;
...
/* myServer specific code up here */
...
/* one might want to make this call as part of the code
 * that overrides the somdRefFromSOMObj method, i.e.
 * in an implementation of a subclass of SOMDServer called
 * myServer
 */
objref =
 myServer_parent_SOMDServer_somdRefFromSOMObj(somSelf, &ev, obj);
```

Original Class

SOMDServer

Related Information

Method: somdObjReferencesCached

somdSOMObjFromRef Method

Purpose

Returns the SOM object corresponding to the specified object reference.

IDL Syntax

SOMObject somdSOMObjFromRef (
in SOMDObject *objref***);**

Description

The **somdSOMObjFromRef** method returns the SOM object associated with the DSOM object reference, *objref*. This method is called by **SOMOA** as part of converting a remote request into a local method call on an object.

Parameters

receiver	A pointer to a **SOMDServer** object.
env	A pointer to the **Environment** structure for the method caller.
objref	A pointer to the DSOM object reference to the SOM object.

Return Value

The **somdSOMObjFromRef** method returns the SOM object associated with the supplied DSOM reference.

Example

```
#include <somd.h>
#include <stack.ih> /* user-generated */

SOMDObject objref;
Environment ev;
SOMObject obj;
...
/* myServer specific code up here */
...
/* one might want to make this call as part of the code
 * that overrides the somdRefFromSOMObj method, i.e.
 * in an implementation of a subclass of SOMDServer called
 * myServer
 */
obj =
myServer_parent_SOMDServer_somdSOMObjFromRef(somSelf,&ev,objref);
```

Original Class

SOMDServer

SOMDServerMgr Class

Description

The **SOMDServerMgr** class provides a programmatic interface to manage server processes. At present, the server processes that can be managed are limited to those present in the Implementation Repository. The choice of Implementation Repository is determined by the environment variable SOMDDIR.

File Stem

servmgr

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

New Methods

somdShutdownServer
somdStartServer
somdRestartServer
somdListServer
somdDisableServer
somdEnableServer
somdIsServerEnabled

somdDisableServer Method

Purpose

Disables a server process from starting until it is explicitly enabled again.

IDL Syntax

ORBStatus somdDisableServer (in string *server_alias***);**

Description

The **somdDisableServer** method disables the server process associated with the server alias. Once a server process has been disabled, it cannot be restarted until it is explicitly enabled again. Initially, all server processes are enabled by default. Note: If the server process to be disabled is currently running, then it is first stopped before disabling. If the method is unsuccessful in stopping the server, the disable method fails.

Parameters

receiver	A pointer to an object of class **SOMDServerMgr.**
ev	A pointer to the **Environment** structure for the calling method.
server_alias	The implementation alias of the server to be disabled.

Return Value

Returns 0 for success or a DSOM error code for failure.

Example

```
#include <somd.h>
#include <servmgr.h>

SOMDServerMgr servmgr;
string  server_alias = ”MyServer”;
ORBStatus rc;
Environment e;

SOM_InitEnvironment(&e);
SOMD_Init(&e);
servmgr = SOMDServerMgrNew();
rc = _somdDisableServer(servmgr, &e, server_alias);
```

Original Class

SOMDServerMgr

Related Information

Methods: somdEnableServer

somdEnableServer Method

Purpose

Enables a server process so that it can be started when required. Initially, all server processes are enabled by default.

IDL Syntax

ORBStatus somdEnableServer (in string *server_alias***);**

Description

The **somdEnableServer** method enables a server process associated with the server alias. Initially, all server processes are enabled by default. Server processes can be disabled by using the **somdDisableServer** method.

Parameters

receiver	A pointer to an object of class **SOMDServerMgr.**
ev	A pointer to the **Environment** structure for the calling method.
server_alias	The implementation alias of the server to be enabled.

Return Value

Returns 0 for success or a DSOM error code for failure.

Example

```
SOMDServerMgr servmgr;
string  server_alias = "MyServer";
ORBStatus rc;
Environment e;

SOM_InitEnvironment(&e);
SOMD_Init(&e);
servmgr = SOMDServerMgrNew();

/* disable the server */
rc = _somdDisableServer(servmgr, &e, server_alias);

/*  do some processing */

/* enable the server */
rc = _somdEnableServer(servmgr, &e, server_alias);
```

Original Class

SOMDServerMgr

Related Information

Methods: somdDisableServer

somdIsServerEnabled Method

Purpose

Determines whether a server process is enabled or not.

IDL Syntax

boolean somdIsServerEnabled (in ImplementationDef *impldef***);**

Description

The **somdIsServerEnabled** method returns a **boolean** corresponding to the current state (enabled/disabled) of the server process.

Parameters

receiver	A pointer to an object of class **SOMDServerMgr.**
ev	A pointer to the **Environment** structure for the calling method.
impldef	A pointer to the **ImplementationDef** object for the server, obtained using the **find_impldef_by_alias** method when it is invoked on the global **SOMD_ImplRepObject**.

Return Value

Returns TRUE if the server is enabled; otherwise, FALSE is returned.

Example

```
#include <somd.h>
#include <servmgr.h>

SOMDServerMgr servmgr;
ImplementationDef impldef;
string  server_alias = "MyServer";
boolean rc;
Environment e;

SOM_InitEnvironment(&e);
SOMD_Init(&e);

impldef = _find_impldef_by_alias(SOMD_ImplRepObject,
                                   &e, server_alias);
servmgr = SOMDServerMgrNew();

/* if server is disabled then enable it*/
if (!_somdIsServerEnabled(servmgr, &e, impldef))
    rc = _somdEnableServer(servmgr, &e, server_alias);
```

Original Class

SOMDServerMgr

Related Information

Methods: somdDisableServer, somdEnableServer

somdListServer Method

Purpose

Queries the state of a server process.

IDL Syntax

ORBStatus somdListServer (in string *server_alias***);**

Description

The **somdListServer** method is invoked to query the status of the server process associated with the server alias. If the server process is running, the return code will be 0 indicating success. Status codes of SOMDERROR_ServerDisabled or SOMDERROR_ServerNotFound may also be returned. The former return code indicates that the server process has been disabled (refer **somdDisableServer**) and the latter indicates that the server process is not currently running.

Parameters

receiver	A pointer to an object of class **SOMDServerMgr.**
ev	A pointer to the **Environment** structure for the calling method.
server_alias	The implementation alias of the server to be listed.

Return Value

Returns 0 if the server process is running; otherwise, a DSOM error code is returned.

Example

```
#include <somd.h>
#include <servmgr.h>

SOMDServerMgr servmgr;
string  server_alias = "MyServer";
ORBStatus rc;
Environment e;

SOM_InitEnvironment(&e);
SOMD_Init(&e);
servmgr = SOMDServerMgrNew();
rc = _somdListServer(servmgr, &e, server_alias);
if (!rc)                        /* server is running */
    rc = _somdShutdownServer(servmgr, &e, server_alias);
else if (rc == SOMDERROR_ServerNotFound)
                            /* server is not running */
    rc = _somdStartServer(servmgr, &e, server_alias);
```

Original Class

SOMDServerMgr

somdRestartServer Method

Purpose

Restarts a server process.

IDL Syntax

ORBStatus somdRestartServer (in string *server_alias***);**

Description

The **somdRestartServer** method is invoked to restart a server process. If the server process currently exists, it will be stopped and started again. If the server process does not exist, a new server process will still be started. If the server process cannot be stopped and/or started for any reason, the method returns a DSOM error code.

Parameters

receiver	A pointer to an object of class **SOMDServerMgr.**
ev	A pointer to the **Environment** structure for the calling method.
server_alias	The implementation alias of the server to be restarted.

Return Value

Returns 0 for success or a DSOM error code for failure.

Example

```
#include <somd.h>
#include <servmgr.h>

SOMDServerMgr servmgr;
string  server_alias = "MyServer";
ORBStatus rc;
Environment e;

SOM_InitEnvironment(&e);
SOMD_Init(&e);
servmgr = SOMDServerMgrNew();
rc = _somdRestartServer(servmgr, &e, server_alias);
```

Original Class

SOMDServerMgr

somdShutdownServer Method

Purpose

Stops a server process.

IDL Syntax

ORBStatus somdShutdownServer (in string *server_alias***);**

Description

The **somdShutdownServer** method is invoked to stop a server process. If the server process corresponding to the server alias exists, it will be stopped and a code indicating success is returned. If the server process does not exist, then the SOMDERROR_ServerNotFound error is returned.

Note: On AIX, this method will fail to stop the server process if the process owner executing this method is not the same as that of either the server process *or* root.

Parameters

receiver	A pointer to an object of class **SOMDServerMgr.**
ev	A pointer to the **Environment** structure for the calling method.
server_alias	The implementation alias of the server to be stopped.

Return Value

Returns 0 for success or a DSOM error code for failure.

Example

```
#include <somd.h>
#include <servmgr.h>

SOMDServerMgr servmgr;
string  server_alias = ”MyServer”;
ORBStatus rc;
Environment e;

SOM_InitEnvironment(&e);
SOMD_Init(&e);
servmgr = SOMDServerMgrNew();
rc = _somdShutdownServer(servmgr, &e, server_alias);
```

Original Class

SOMDServerMgr

somdStartServer Method

Purpose

Starts a server process.

IDL Syntax

ORBStatus somdStartServer (in string *server_alias***);**

Description

The **somdStartServer** method is invoked to start a server process. If the server process does not exist, the server process is started and the code indicating success is returned. If the server process already exists, then the return code will still indicate success and the server process will be undisturbed.

Parameters

receiver	A pointer to an object of class **SOMDServerMgr.**
ev	A pointer to the **Environment** structure for the calling method.
server_alias	The implementation alias of the server to be started.

Return Value

Returns 0 for success or a DSOM error code for failure.

Example

```
#include <somd.h>
#include <servmgr.h>

SOMDServerMgr servmgr;
string  server_alias = ”MyServer”;
ORBStatus rc;
Environment e;

SOM_InitEnvironment(&e);
SOMD_Init(&e);
servmgr = SOMDServerMgrNew();
rc = _somdStartServer(servmgr, &e, server_alias);
```

Original Class

SOMDServerMgr

SOMOA Class

Description

The **SOMOA** class is DSOM's basic object adapter. **SOMOA** is a subclass of the abstract **BOA** class, and provides implementations of all the **BOA** methods. The **SOMOA** class also introduces methods for receiving and dispatching requests on SOM objects. **SOMOA** provides some additional methods for creating and managing object references.

File Stem

somoa

Base

BOA

Metaclass

SOMMSingleInstance

Ancestor Classes

BOA, **SOMObject**

New Methods

activate_impl_failed*
change_id*
create_constant*
create_SOM_ref*
execute_next_request*
execute_request_loop*
get_SOM_object*

(* This class and its methods were added by DSOM to supplement the published CORBA 1.1 interfaces.)

Overridden Methods

change_implementation
create
deactivate_impl
deactivate_obj
dispose
get_id
get_principal
impl_is_ready
obj_is_ready
set_exception
somInit
somUninit

activate_impl_failed Method

Purpose

Sends a message to the DSOM daemon indicating that a server did not activate.

IDL Syntax

void activate_impl_failed (
in ImplementationDef *implDef*,
in long *rc*);

Description

The **activate_impl_failed** method sends a message to the DSOM daemon (**somdd**) indicating that the server did not activate.

Parameters

receiver	A pointer to the **SOMOA** object that attempted to activate the implementation.
env	A pointer to the **Environment** structure for the method caller.
implDef	A pointer to the **ImplementationDef** object representing the implementation that failed to activate.
rc	A return code designating the reason for failure.

Return Value

None.

Example

```
#include <somd.h> /* needed by all servers */
main(int argc, char **argv)
{
   Environment ev;
   SOM_InitEnvironment(&ev);

   /* Initialize the DSOM run-time environment */
   SOMD_Init(&ev);

   /* Retrieve its ImplementationDef from the Implementation
      Repository by passing its implementation ID as a key */
   SOMD_ImplDefObject =
      _find_impldef(SOMD_ImplRepObject, &ev, argv[1]);

   /* create the SOMOA */
   SOMD_SOMOAObject = SOMOANew();
...
/* suppose something went wrong with server initialization */
...
/* tell the daemon (via SOMOA) that activation failed */
_activate_impl_failed(SOMD_SOMOAObject,
                      &ev, SOMD_ImplDefObject, rc);
```

Original Class

SOMOA

change_id Method

Purpose

Changes the reference data associated with an object.

IDL Syntax

void change_id (
in SOMDObject *objref,*
in ReferenceData *id* **);**

Description

The **change_id** changes the **ReferenceData** associated with the object identified by *objref.* The **ReferenceData** previously stored in the **SOMOA**'s reference data table is replaced with the value of *id*. The new ID cannot be larger than the maximum size of the original **ReferenceData** (usually specified as 1024 bytes).

Parameters

receiver	A pointer to the **SOMOA** object managing the implementation.
env	A pointer to the **Environment** structure for the method caller.
objref	A pointer to the **SOMDObject** which identifies the object.
id	A pointer to the **ReferenceData** structure representing the object to be created.

Return Value

None.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>

SOMDObject objref;
ReferenceData id1, id2;
InterfaceDef intfdef;
...
objref = _create(SOMD_SOMOAObject,&ev, id1,
                 intfdef, SOMD_ImplDefObject);
...
/* change the ReferenceData associated with a SOMDObject */
_change_id(SOMD_SOMOAObject, &ev, objref, id2);
```

create_constant Method

Purpose

Creates a “constant” object reference.

IDL Syntax

SOMDObject create_constant (
in ReferenceData *id***,**
in InterfaceDef *intf*,
in ImplementationDef *impl***);**

Description

The **create_constant** method is a variant of the **create** method. Like **create**, it creates an object reference for an object with the specified interface and associates the supplied **ReferenceData** with the object reference. The **ReferenceData** can later be retrieved using the **get_id** method. Unlike **create**, this method creates a “constant” reference whose ID value cannot be changed. (See the **change_id** method of **SOMOA**.) This is because the ID is maintained as a constant part of the object reference state, versus stored in the reference data table for the server.

This method would be used whenever the application prefers not to maintain an object’s **ReferenceData** in the server’s reference data table.

Parameters

receiver	A pointer to the **SOMOA** object managing the implementation.
env	A pointer to the **Environment** structure for the method caller.
id	A pointer to the **ReferenceData** structure containing application-specific information describing the target object.
intf	A pointer to the **InterfaceDef** object which describes the interface of the target object.
impl	A pointer to the **ImplementationDef** object which describes the application (server) process which implements the target object.

Return Value

The **create_constant** method returns a pointer to a **SOMDObject**. *Ownership* of the new object reference is transferred to the caller.

Example

```
#include <somd.h>
#include <repostry.h>
#include <intfacdf.h>

Environment ev;
ReferenceData id;
InterfaceDef intfdef;
SOMDObject objref;
string fname; /* file name to be saved with reference */
...
/* create the id for the reference */
id._maximum = id._length = strlen(fname)+1;
id._buffer = (string) SOMMalloc(strlen(fname)+1);
strcpy(id._buffer,fname);

/* get the interface def object for interface Foo*/
intfdef = _lookup_id(SOM_InterfaceRepository, &ev, ”Foo”);

objref = _create_constant(SOMD_SOMOAObject,
                 &ev, id, intfdef, SOMD_ImplDefObject);
```

Original Class

SOMOA

Related Information

Methods: create, create_SOM_ref, dispose, get_id, is_constant

create_SOM_ref Method

Purpose

Creates a simple, transient DSOM reference to a SOM object.

IDL Syntax

SOMDObject create_SOM_ref (
in SOMObject *somobj,*
in ImplementationDef *impl***);**

Description

The **create_SOM_ref** method creates a simple DSOM reference (**SOMDObject**) for a local SOM object. The reference is “special” in that there is no explicit **ReferenceData** associated with the object. Also, this object reference is only valid while the target SOM object exists.

The **SOMObject** associated with the SOM_ref can be retrieved via the **get_SOM_object** method. The **is_SOM_ref** method of **SOMDObject** can be used to determine whether the reference was created using **create_SOM_ref** or not.

Parameters

receiver	A pointer to the **SOMOA** object managing the implementation.
env	A pointer to the **Environment** structure for the method caller.
somobj	A pointer to the local **SOMObject** to be referenced.
impl	A pointer to the **ImplementationDef** of the calling server process.

Return Value

The **create_SOM_ref** method returns a pointer to a **SOMDObject**. *Ownership* of the new object reference is transferred to the caller.

Example

```
#include <somd.h>

SOMDObject objref;
Environment ev;
SOMObject obj;
...
/* one might want to make this call as part of the code
 * that overrides the somdRefFromSOMObj method, i.e.
 * in an implementation of a subclass of SOMDServer.
 */
objref = _create_SOM_ref(SOMD_SOMOAObject, &ev, obj);
```

Original Class

SOMOA

Related Information

Methods: get_SOM_object, is_SOM_ref

execute_next_request Method

Purpose

Receives a request message, executes the request, and returns to the caller.

IDL Syntax

ORBStatus execute_next_request (
in Flags *waitFlag* **);**

Description

The **execute_next_request** method receives the next request message, executes the request, and sends the result to the caller.

If the server's **ImplementationDef** indicates the server is multi-threaded (the **impl_flags** has the IMPLDEF_MULTI_THREAD flag set), each request will be run by **SOMOA** in a separate thread (OS/2 only).

Parameters

receiver	A pointer to the **SOMOA** object managing the implementation.
env	A pointer to the **Environment** structure for the method caller.
waitFlag	A **Flags** value (unsigned long) indicating whether the method should block if there is no message pending (SOMD_WAIT) or return with an error (SOMD_NO_WAIT).

Return Value

The **execute_next_request** method returns an **ORBStatus** value representing the return value for the operation. SOMDERROR_NoMessages is returned if the method is invoked with SOMD_NO_WAIT and no message is available.

Example

```
#include <somd.h>

/* server initialization code ... */
SOM_InitEnvironment(&ev);

/* signal DSOM that server is ready */
_impl_is_ready(SOMD_SOMOAObject, &ev, SOMD_ImplDefObject);

while (ev._major == NO_EXCEPTION) {
     (void) _execute_next_request(SOMD_SOMOAObject,&ev,SOMD_WAIT);
     /* perform appl-specific code between messages here, e.g.,*/
     numMessagesProcessed++;
}
```

Original Class

SOMOA

Related Information

Methods: execute_request_loop

execute_request_loop Method

Purpose

Receives a request message, executes the request, and returns the result to the calling client.

IDL Syntax

ORBStatus execute_request_loop (
in Flags *waitFlag***);**

Description

The **execute_request_loop** method initiates a loop that waits for a request message, executes the request, and returns the result to the client who invoked the request. When called with the SOMD_WAIT flag, this method loops infinitely (or until an error). When called with the SOMD_NO_WAIT flag, this method loops as long as it finds a request message to process.

The SOMD_NO_WAIT flag is useful when writing event-driven applications where there are event sources other than DSOM requests (for example, user input). In this case, DSOM cannot be given exclusive control. Instead, a DSOM event handler can be written using the SOMD_NO_WAIT option, to process all pending requests before returning control to the event manager.

If the server's **ImplementationDef** indicates the server is multi-threaded (the **impl_flags** has the IMPLDEF_MULTI_THREAD flag set), each request will be run by **SOMOA** in a separate thread (OS/2 only).

Parameters

receiver	A pointer to the **SOMOA** object managing the implementation.
env	A pointer to the **Environment** structure for the method caller.
waitFlag	A **Flags** bitmask (unsigned long) indicating whether the method should block (SOMD_WAIT) or return to the caller (SOMD_NO_WAIT) when there is no request message pending.

Return Value

The **execute_request_loop** method may return an **OBJ_ADAPTER** exception which contains an DSOM error code for the operation. SOMDERROR_NoMessages is returned as an **ORBStatus** code if the method is invoked with SOMD_NO_WAIT and no message is pending.

Example

```
#include <somd.h>

/* server initialization code ... */
...
_impl_is_ready(SOMD_SOMOAObject, &ev, SOMD_ImplDefObject);

/* turn control over to SOMOA */
(void) _execute_request_loop(SOMD_SOMOAObject, &ev, SOMD_WAIT);
```

Original Class

SOMOA

Related Information

Functions: SOMD_RegisterCallback
Methods: execute_next_request

See Chapter 12 of the *SOM Toolkit User's Guide* for a description of the Event Management (EMan) framework, for writing event-driven applications.

get_SOM_object Method

Purpose

Get the SOM object associated with a simple DSOM reference.

IDL Syntax

SOMObject get_SOM_object (
in SOMDObject *somref***);**

Description

The **get_SOM_object** method returns the SOM object associated with a reference created by the **create_SOM_ref** method.

Parameters

receiver	A pointer to the **SOMOA** object managing the implementation.
env	A pointer to the **Environment** structure for the method caller.
somref	A pointer to a **SOMDObject** created by the **create_SOM_ref** method.

Return Value

The **get_SOM_object** method returns the SOM object associated with the reference.

Example

```
#include <somd.h>

SOMDObject objref;
Environment ev;
SOMObject obj;
...
if (_is_SOM_ref(objref, &ev))
    /* we know objref is a simple reference, so we can ... */
    obj = _get_SOM_object(SOMD_SOMOAObject, &ev, objref);
...
```

Original Class

SOMOA

Related Information

Methods: create_SOM_ref, is_SOM_ref