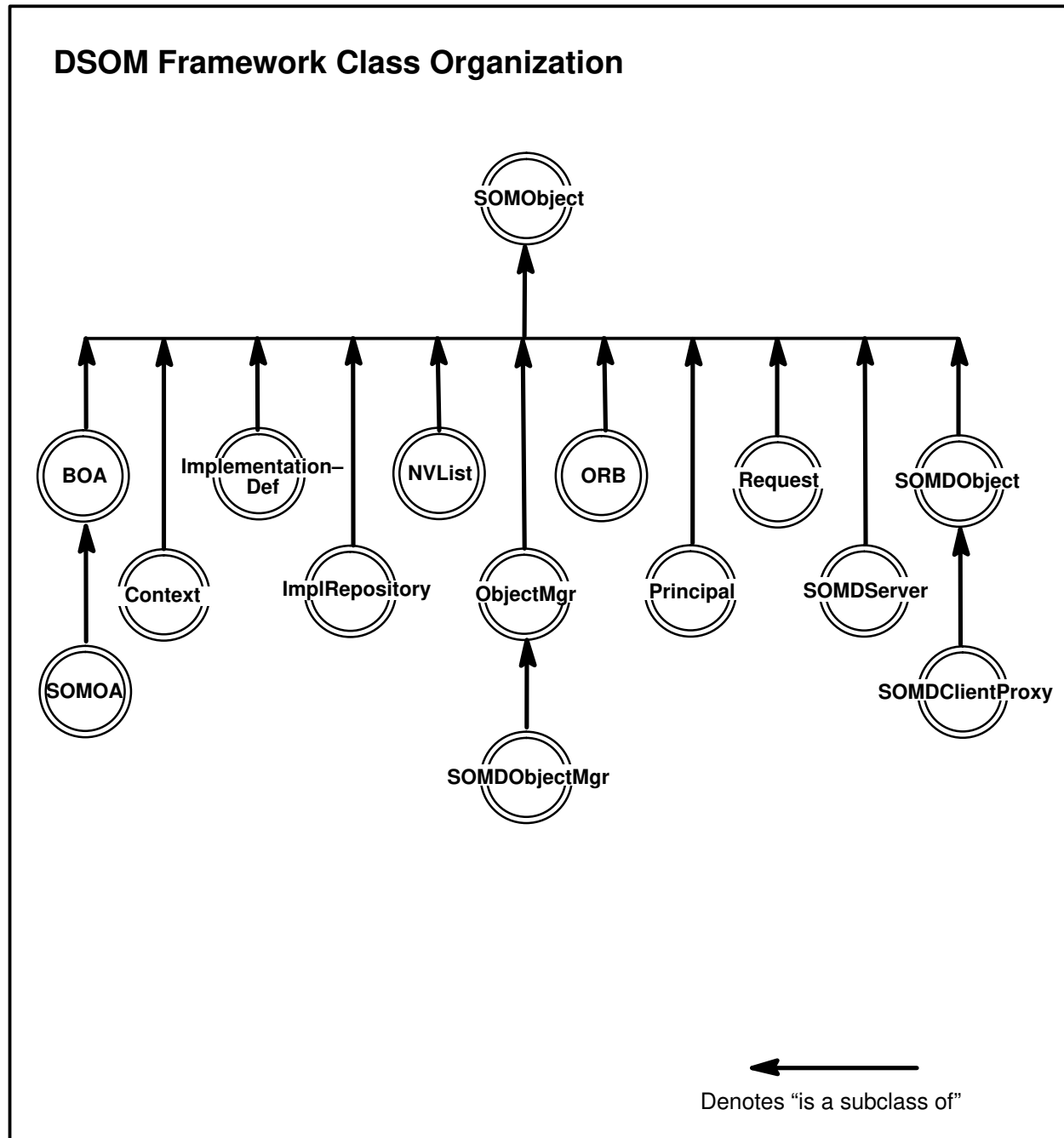


## DSOM Framework Reference



---

## 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 (e.g., **ImplementationDef**, **Principal**)
- CORBA 1.1 did not address the function specified by the interface (e.g., “lifecycle” services for object creation and deletion)
- 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 Chapter 6 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**

<i>ptr</i>	A pointer to memory that has been dynamically allocated by DSOM for a method return value or <b>out</b> 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**

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

<i>reqs</i>	The address of an array of <b>Requests</b> objects which are to be initiated in parallel.
<i>env</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>count</i>	The number of <b>Request</b> objects in <i>reqs</i> .
<i>invoke_flags</i>	A <b>Flags</b> (unsigned long) value, used to indicate the following options: <ul style="list-style-type: none"> <li>INV_NO_RESPONSE – Indicates the caller does not intend to get any results or <b>out</b> parameter values from any of the requests. The requests can be treated as if they are <b>oneway</b> operations.</li> <li>INV_TERM_ON_ERR – If one of the requests causes an error, the remaining requests are not sent.</li> </ul>

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 Chapter 6 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.

## **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.

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 method 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\_RegisterCallback Function

### Purpose

Registers a callback function for handling DSOM request events.

### C Syntax

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

### Description

When writing event-driven applications where there are event sources other than DSOM requests (e.g., user input, mouse clicks, etc.), 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 Manager (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**.

**Note:** the function **SOMD\_RegisterCallback** must be declared with “system linkage” on OS/2.

### Parameters

<i>emanObj</i>	A pointer to an instance of <b>SOMEEMan</b> , the Event Manager object.
<i>func</i>	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 <b>EMRegProc</b> type defined in “eman.h”):

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

void func (SOMEEvent event, void *eventData)
```

### Return Value

None.



## Example

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

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

void SOMD_RegisterCallback (SOMEEMan emanObj, EMRegProc *func);

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

    _execute_request_loop (SOMD_SOMOAObject, &ev, SOMD_NO_WAIT);
}

main()
{
    ...
    eman = SOMEEManNew();
    SOMD_RegisterCallback(eman, DSOMEEventCallBack);

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

## Related Information

See Chapter 12 of the *SOM Toolkit User's Guide* for a description of the Event Manager (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 (shared memory segments, semaphores, etc.) allocated to the calling process for use by DSOM. This function should be called before a process exits, to ensure system resources are reused.

### **Parameters**

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

---

## 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 (i.e., 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

<i>receiver</i>	A pointer to a <b>BOA (SOMOA)</b> object for the server.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	A pointer to the <b>SOMDObject</b> object which refers to the application object whose implementation is to be changed.
<i>impl</i>	A pointer to the <b>ImplementationDef</b> 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 (i.e., 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

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

## BOA class

### 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

<i>receiver</i>	A pointer to a <b>BOA (SOMOA)</b> object for the server.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>impl</i>	A pointer to the <b>ImplementationDef</b> 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 (i.e., 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

<i>receiver</i>	A pointer to a <b>BOA (SOMOA)</b> object for the server.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	A pointer to a <b>SOMDObject</b> 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

<i>receiver</i>	A pointer to a <b>BOA</b> object for the server.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	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

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

### 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

<i>receiver</i>	A pointer to a <b>BOA (SOMOA)</b> object for the server.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	A pointer to the object reference which is the target of the method call.
<i>req_ev</i>	A pointer to the <b>Environment</b> 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(SOMDObject *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

<i>receiver</i>	A pointer to a <b>BOA (SOMOA)</b> object for the server.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>impl</i>	A pointer to the <b>ImplementationDef</b> 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 (i.e., 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

<i>receiver</i>	A pointer to a <b>BOA (SOMOA)</b> object for the server.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	A pointer to a <b>SOMDObject</b> which identifies the object (server) which is ready.
<i>impl</i>	A pointer to the <b>ImplementationDef</b> 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

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

<i>receiver</i>	A pointer to the <b>Context</b> object for which a child is to be created.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>ctx_name</i>	The name of the child <b>Context</b> to be created.
<i>child_ctx</i>	The address where a pointer to the created child <b>Context</b> 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

<i>receiver</i>	A pointer to the <b>Context</b> object from which values will be deleted.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>prop_name</i>	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**

<i>receiver</i>	A pointer to the <b>Context</b> object to be deleted.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>del_flag</i>	A bitmask (unsigned long). If the option flag CTX_DELETE_DESCENDENTS is specified, the method deletes the indicated <b>Context</b> object and all of its descendent <b>Context</b> 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

<i>receiver</i>	A pointer to the <b>Context</b> object from which the properties are to be retrieved.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>start_scope</i>	An <b>Identifier</b> specifying the name of the <b>Context</b> object at which search for the properties should commence.
<i>op_flags</i>	A bitmask ( <b>long</b> ). The operation flag CTX_RESTRICT_SCOPE may be specified. Searching is limited to the specified search scope or <b>Context</b> object.
<i>prop_name</i>	An <b>Identifier</b> specifying the name of the property value(s) to return.
<i>values</i>	The address to store a pointer to the resulting <b>NVList</b> object.

### Return Value

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

## Context class

### 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

<i>receiver</i>	A pointer to the <b>Context</b> object to which the value is to be added.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>prop_name</i>	The name of the property to be added.
<i>value</i>	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

<i>receiver</i>	A pointer to the <b>Context</b> object for which the properties are to be set.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>values</i>	A pointer to an <b>NVList</b> object containing the properties to be set.

### 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).

**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 **ImplementationDefs** are limited to 255 bytes. Implementations aliases used in **ImplementationDefs** are limited to 50 bytes. Class names used in **ImplementationDefs** 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\_classes\_by\_impldef  
find\_impldef  
find\_impldef\_by\_alias  
find\_impldef\_by\_class  
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

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>implid</i>	The <b>ImplId</b> identifier for the <b>ImplementationDef</b> of the desired server.
<i>classname</i>	A <b>string</b> 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

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>impldef</i>	A pointer to the <b>ImplementationDef</b> 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

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>implid</i>	The <b>ImplId</b> 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\_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

A sequence of strings is returned, representing a list of the class names associated with the server implementation identified by *implid*.

### Parameters

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>implid</i>	The <b>ImplId</b> 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

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>implid</i>	The <b>ImplId</b> of the desired <b>ImplementationDef</b> .

### 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

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>alias_name</i>	User-friendly name used to identify the implementation.

### 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>

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\_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

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>classname</i>	A <b>string</b> 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\_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

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>implid</i>	A pointer to an <b>ImplRepository</b> object.
<i>classname</i>	A <b>string</b> 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

<i>receiver</i>	A pointer to the <b>ImplRepository</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>impldef</i>	A pointer to an <b>ImplementationDef</b> 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. Addition 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

somlnit

## 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

<i>receiver</i>	A pointer to the <b>NVList</b> object to which the item will be added.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>item_name</i>	The name of the item to be added.
<i>item_type</i>	The data type of the item to be added.
<i>value</i>	A pointer to the value of the item to be added.
<i>value_len</i>	The length of the item value to be added.
<i>item_flags</i>	A <b>Flags</b> bitmask (unsigned long). The <i>item_flags</i> 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**

<i>receiver</i>	A pointer to the <b>NVList</b> object to be freed.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to the <b>NVList</b> object whose out-arg memory is to be freed.
<i>env</i>	A pointer to the <b>Environment</b> 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 */
}
...
```

**NVList class**

## **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

<i>receiver</i>	A pointer to the <b>NVList</b> object on which count is desired.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>count</i>	A pointer to where the method will store the <b>long</b> 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

<i>receiver</i>	A pointer to an <b>NVList</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>item_number</i>	The position (index) of the item in the list. The <i>item_number</i> ranges from 0 to $n-1$ , where $n$ is the total number of items in the list.
<i>item_name</i>	A pointer to where the name of the item should be returned.
<i>item_type</i>	A pointer to where the data type of the item should be returned.
<i>value</i>	A pointer to where a pointer to the value of the item should be returned.
<i>value_len</i>	A pointer to where the length of the item value should be returned.
<i>item_flags</i>	A <b>Flags</b> bitmask (unsigned long). The <i>item_flags</i> 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

<i>receiver</i>	A pointer to an <b>NVList</b> which contains the item to be set.										
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.										
<i>item_number</i>	The position (index) of the item in the list. The <i>item_number</i> ranges from 0 to $n-1$ , where $n$ is the total number of items in the list.										
<i>item_name</i>	The name of the set item.										
<i>item_type</i>	The data type of the set item.										
<i>value</i>	A pointer to the value of the set item.										
<i>value_len</i>	The length of the set item value.										
<i>item_flags</i>	A <b>Flags</b> bitmask (unsigned long). The <i>item_flags</i> can be one of the following values to indicate parameter direction: <table data-bbox="537 1276 1146 1373"> <tr> <td>ARG_IN</td><td>The argument is input only.</td></tr> <tr> <td>ARG_OUT</td><td>The argument is output only.</td></tr> <tr> <td>ARG_INOUT</td><td>The argument is input/output.</td></tr> </table> <p>In addition, <i>item_flags</i> may also contain the following values:</p> <table data-bbox="537 1467 1443 1593"> <tr> <td>IN_COPY_VALUE</td><td>Indicates an internal copy of the argument is made and used.</td></tr> <tr> <td>DEPENDENT_LIST</td><td>Indicates that a specified sublist must be freed when the parent list is freed.</td></tr> </table>	ARG_IN	The argument is input only.	ARG_OUT	The argument is output only.	ARG_INOUT	The argument is input/output.	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.
ARG_IN	The argument is input only.										
ARG_OUT	The argument is output only.										
ARG_INOUT	The argument is input/output.										
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 (i.e., 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

Indicates that the client wants the object destroyed.

### 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 **SOMObjectMgr** forwards the deletion request to the remote server, and then frees the local proxy object.

### Parameters

<i>receiver</i>	A pointer to an <b>ObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	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.

### Parameters

<i>receiver</i>	A pointer to an <b>ObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	A pointer to the 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 "SOMObject 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

<i>receiver</i>	A pointer to an <b>ObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>id</i>	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

<i>receiver</i>	A pointer to an <b>ObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>objclass</i>	An <b>Identifier</b> representing the type of the new object.
<i>hints</i>	A <b>string</b> 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

<i>receiver</i>	A pointer to an <b>ObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	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 **Requests**.

*Ownership* of the allocated *new\_list* is transferred to the caller.

### Parameters

<i>receiver</i>	A pointer to the <b>ORB</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>count</i>	An integer representing the number of elements to allocate for the list.
<i>new_list</i>	A pointer to the address where the method will store a pointer to the allocated <b>NVList</b> 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 **Requests** invoking that operation.

### Parameters

<i>receiver</i>	A pointer to the <b>ORB</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>oper</i>	A pointer to the <b>OperationDef</b> object representing the operation for which the NVList is to be initialized.
<i>new_list</i>	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

<i>receiver</i>	A pointer to the <b>ORB</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>ctx</i>	A pointer to where the method will store a pointer to the returned <b>Context</b> 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 Chapter 6 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.

### Parameters

<i>receiver</i>	A pointer to the <b>ORB</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>obj</i>	A pointer to a <b>SOMDObject</b> 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

<i>receiver</i>	A pointer to the <b>ORB</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>str</i>	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

<i>receiver</i>	A pointer to a <b>Request</b> object.	
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.	
<i>name</i>	An identifier representing the name of the argument to be added.	
<i>arg_type</i>	The typecode for the argument to be added.	
<i>value</i>	A pointer to the argument value to be added.	
<i>len</i>	The length of the argument.	
<i>arg_flags</i>	A <b>Flags</b> bitmask (unsigned long). The <i>arg_flags</i> 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, <i>arg_flags</i> 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**

<i>receiver</i>	A pointer to a <b>Request</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to a <b>Request</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>response_flags</i>	A <b>Flags</b> bitmask (unsigned long) containing control information for the <b>get_response</b> method. The <i>response_flags</i> 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, 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

<i>receiver</i>	A pointer to a <b>Request</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>invoke_flags</i>	<b>AFlags</b> bitmask (unsigned long) representing control information for the <b>invoke</b> method. There are currently no flags defined for the <b>invoke</b> 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**, **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** method.

### Parameters

<i>receiver</i>	A pointer to a <b>Request</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>invoke_flags</i>	A <b>Flags</b> bitmask (unsigned long) containing <b>send</b> method control information. The argument <i>invoke_flags</i> 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 ( <b>inout</b> or <b>out</b> ) 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

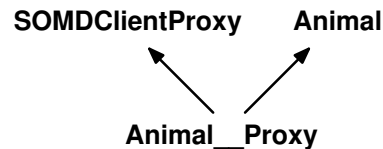
Methods: `invoke`, `get_response`, `delete`

---

## 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\***

(\* 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

<i>receiver</i>	A pointer to the <b>SOMDClientProxy</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to the <b>SOMDClientProxy</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to the <b>SOMDClientProxy</b> object for the desired remote target object.
<i>env</i>	A pointer to the <b>Environment</b> 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**

## 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

<i>receiver</i>	A pointer to the <b>SOMDClientProxy</b> object for the desired remote target object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to the <b>SOMDClientProxy</b> object for the desired remote target object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to the <b>SOMDClientProxy</b> object for the desired remote target object.
<i>env</i>	A pointer to the <b>Environment</b> 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: e.g., **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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.		
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.		
<i>ctx</i>	A pointer to the <b>Context</b> object of the requested operation.		
<i>operation</i>	The name of the operation to be performed on the target object, <i>receiver</i> .		
<i>arg_list</i>	A pointer to a list of arguments ( <b>NVList</b> ). If this argument is NULL, the argument list can be assembled by repeated calls to the <b>add_arg</b> method on the <b>Request</b> object created by calling this method.		
<i>result</i>	A pointer to a <b>NamedValue</b> structure where the result of applying <i>operation</i> to <i>receiver</i> should be stored.		
<i>request</i>	A pointer to storage for the address of the created <b>Request</b> object.		
<i>req_flags</i>	A <b>Flags</b> bitmask (unsigned long) that may contain the following flag value: <table border="0" data-bbox="535 1417 1443 1608"> <tr> <td>OUT_LIST_MEMORY</td><td>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 <b>create_request</b> call.</td></tr> </table>	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 <b>create_request</b> call.
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 <b>create_request</b> 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);

```

**SOMDObject class**

## **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

<i>receiver</i>	A pointer to the <b>SOMDObject</b> object to create the request.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>operation</i>	The Identifier of the operation for which the argument list is being created.
<i>arg_list</i>	A pointer to the location where the method will store a pointer to the resulting argument list.
<i>result</i>	A pointer to the <b>NamedValue</b> structure which will be used to hold the result. The <i>result</i> '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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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 (i.e., 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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.

### Return Value

The **is\_constant** method returns TRUE if the object reference was generated by **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\_nil**, **is\_proxy**, **is\_SOM\_ref**

## 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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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\_nil**, **is\_constant**

## 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

<i>receiver</i>	A pointer to a <b>SOMDObject</b> object.
<i>env</i>	A pointer to the <b>Environment</b> 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**

---

## 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**

### 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

<i>receiver</i>	A pointer to a <b>SOMDObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>objclass</i>	An <b>Identifier</b> 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.

### 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

<i>receiver</i>	A pointer to a <b>SOMDObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>serverid</i>	An <b>ImplId</b> 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

<i>receiver</i>	A pointer to a <b>SOMDObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>servername</i>	An <b>string</b> which 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

<i>receiver</i>	A pointer to a <b>SOMDObjectMgr</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>objclass</i>	An <b>Identifier</b> 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 (**SOMDObjects**) and SOM objects, and dispatching methods on objects.

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

### File Stem

**SOMDServer**

### 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

<i>receiver</i>	A pointer to a <b>SOMDServer</b> object capable of creating a instance of the specified class.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>objclass</i>	The class of the object for which an instance is to be created.
<i>hints</i>	A <b>string</b> 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

<i>receiver</i>	A pointer to a <b>SOMDServer</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>somobj</i>	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

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

<i>receiver</i>	A pointer to a <b>SOMDServer</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>objclass</i>	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.

### 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

<i>receiver</i>	A pointer to an object of class <b>SOMDServer</b> .
<i>ev</i>	A pointer to the <b>Environment</b> 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

<i>receiver</i>	A pointer to a <b>SOMDServer</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>somobj</i>	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

<i>receiver</i>	A pointer to a <b>SOMDServer</b> object.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>objref</i>	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**

---

## 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

<i>receiver</i>	A pointer to the <b>SOMOA</b> object that attempted to activate the implementation.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>implDef</i>	A pointer to the <b>ImplementationDef</b> object representing the implementation that failed to activate.
<i>rc</i>	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

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

### Return Value

None.

### 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);
```

## 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 “contant” 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

<i>receiver</i>	A pointer to the <b>SOMOA</b> object managing the implementation.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>id</i>	A pointer to the <b>ReferenceData</b> structure containing application-specific information describing the target object.
<i>intf</i>	A pointer to the <b>InterfaceDef</b> object which describes the interface of the target object.
<i>impl</i>	A pointer to the <b>ImplementationDef</b> 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.

## SOMOA class

### 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 tell if the reference was created using **create\_SOM\_ref** or not.

### Parameters

<i>receiver</i>	A pointer to the <b>SOMOA</b> object managing the implementation.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>somobj</i>	A pointer to the local <b>SOMObject</b> to be referenced.
<i>impl</i>	A pointer to the <b>ImplementationDef</b> 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

Receive a request message, execute the request, and return 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 multithreaded (the **impl\_flags** has the **IMPLDEF\_MULTI\_THREAD** flag set), each request will be run by **SOMOA** in a separate thread.

### Parameters

<i>receiver</i>	A pointer to the <b>SOMOA</b> object managing the implementation.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>waitFlag</i>	A <b>Flags</b> value (unsigned long) indicating whether the method should block if there is no message pending ( <b>SOMD_WAIT</b> ) or return with an error ( <b>SOMD_NO_WAIT</b> ).

### 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 (e.g., user input, etc.). 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 multithreaded (the **impl\_flags** has the **IMPLDEF\_MULTI\_THREAD** flag set), each request will be run by **SOMOA** in a separate thread.

### Parameters

<i>receiver</i>	A pointer to the <b>SOMOA</b> object managing the implementation.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>waitFlag</i>	A <b>Flags</b> bitmask (unsigned long) indicating whether the method should block ( <b>SOMD_WAIT</b> ) or return to the caller ( <b>SOMD_NO_WAIT</b> ) 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**

**SOMOA class**

## **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 Manager (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

<i>receiver</i>	A pointer to the <b>SOMOA</b> object managing the implementation.
<i>env</i>	A pointer to the <b>Environment</b> structure for the method caller.
<i>somref</i>	A pointer to a <b>SOMDObject</b> created by the <b>create_SOM_ref</b> 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**

