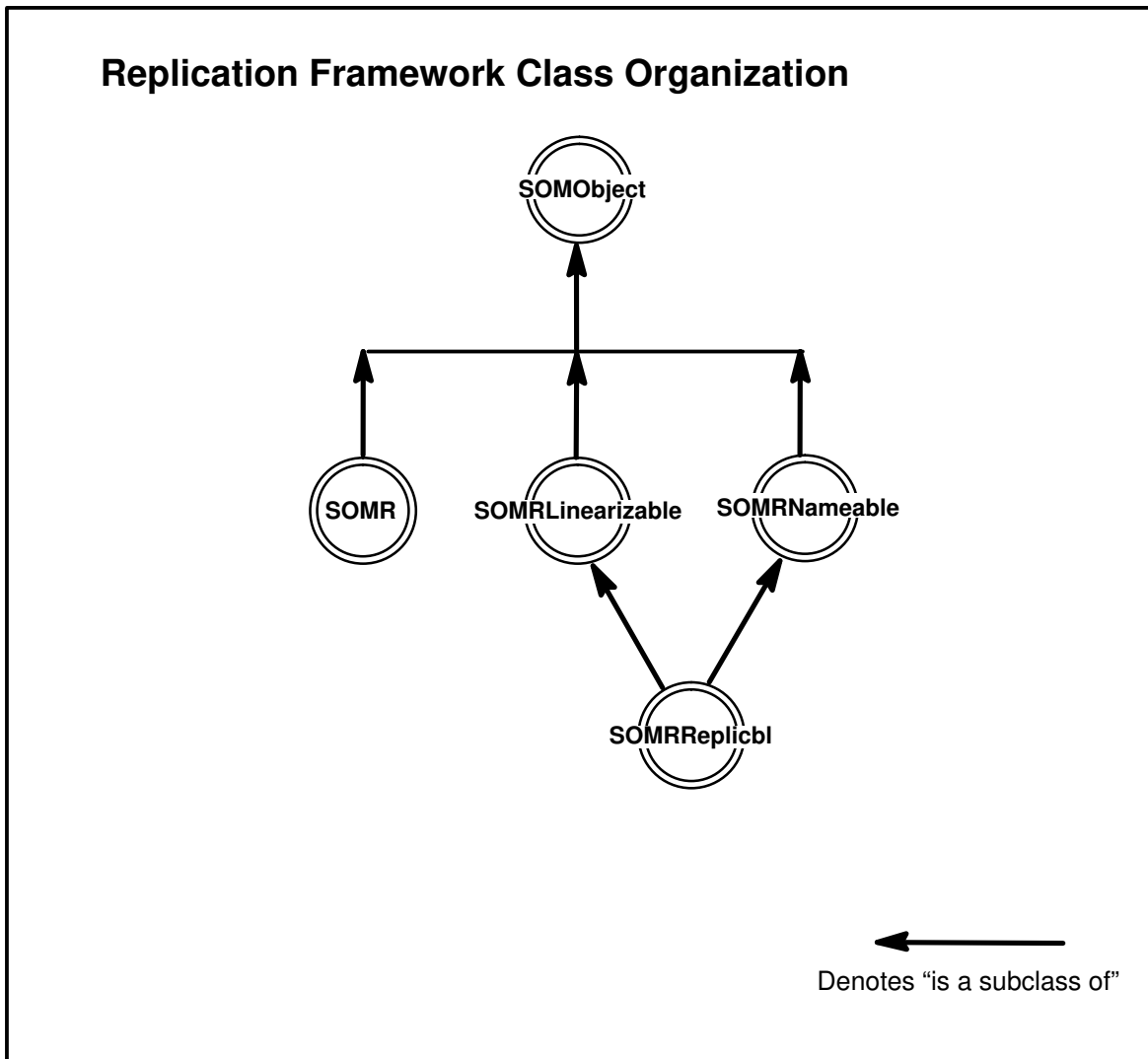


# Replication Framework Reference



---

## SOMR Class

### Description

The **SOMR** class creates and initializes several manager objects required by the Replication Framework. To use the Replication Framework, an application program must create a single instance of the **SOMR** class at the beginning of the application.

### File Stem

somr

### Base Classes

SOMObject

### Metaclass

SOMMSingleInstance

### Ancestor Classes

SOMObject

### New Methods

None.

### Overriding Methods

somlnit

---

## SOMRLinearizable Class

### Description

The **SOMRLinearizable** class provides an interface for objects that are required to be copied. Two methods are introduced: **somrGetState** and **somrSetState**. The **somrGetState** method encodes the object into a byte string. The **somrSetState** method restores the value of the object from the byte string. Currently, only an interface is provided by this class; thus, these methods must be overridden.

### File Stem

linear

### Base Classes

SOMObject

### Metaclass

SOMClass

### Ancestor Classes

SOMObject

### New Methods

somrGetState  
somrSetState

### Overriding Methods

somlInit

## somrGetState Method

### Purpose

Converts the internal state of an object into a byte string.

### IDL Syntax

```
void somrGetState (  
    inout string buf);
```

### Description

The **somrGetState** method converts the internal state of an object into a byte string and returns a pointer to the string. (The length of the string is in the first **sizeof(long)** bytes of this string. Although typed as a string, the data area following the length field may contain NULLs.) The implementor must allocate the necessary memory for the string.

The **somrGetState** method must be overridden in a subclass.

The ownership of this string is transferred to the caller of this method.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRLinearizable</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>buf</i>	A pointer to a buffer where the outgoing byte string will be placed.

### Return Values

None.

### Examples

Suppose that a class `MyObj` has a single instance variable `mydata`, which is a pointer to a string. Use an implementation such as the following to override the **somrGetState** method:

```
SOM_Scope void SOMLINK somrGetState( MyObj somSelf, Environment *Env, string *buf )  
{  
    long len;  
    MyObjData *somThis = MyObjGetData( somSelf );  
    *buf = SOMMalloc( len = strlen( _mydata ) + 1 + sizeof(long) );  
    strcpy( *buf + sizeof(long), _mydata );  
    *(long*)buf = len;  
}
```

### Original Class

**SOMRLinearizable**

### Related Information

Methods: **somrSetState**, **somrApplyUpdates**

## somrSetState Method

### Purpose

Converts a given linear byte string into its internal state.

### IDL Syntax

```
void somrSetState (
    in string buf);
```

### Description

The **somrSetState** method is the reverse of **somrGetState**. The **somrSetState** method converts the given byte string into its internal state. (The length of the string is in the first **sizeof(long)** bytes of this string. Although typed as a string, the data area following the length field may contain NULLs.)

This method must be overridden in a subclass.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRLinearizable</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>buf</i>	A pointer to the incoming byte string to be converted.

### Return Values

None.

### Examples

Suppose that a class `MyObj` has a single instance variable `mydata`, which is a pointer to a string. Use an implementation such as the following to override the **somrSetState** method:

```
SOM_Scope void SOMLINK somrSetState( MyObj somSelf, Environment *Env, string buf )
{
    MyObjData *somThis = MyObjGetData( somSelf );
    _mydata = strcpy( SOMMalloc( *(long*)buf ), buf + sizeof(long) );
}
```

### Original Class

**SOMRLinearizable**

### Related Information

Methods: **somrGetState**, **somrApplyUpdates**

---

## SOMRNameable Class

### Description

The **SOMRNameable** class provides an interface for objects that require a string name. Two methods are introduced: **somrGetObjName** and **somrSetObjName**. The **somrGetObjName** method returns a pointer to the object's name. The **somrSetObjName** method sets the object's name pointer to point to the given string.

### File Stem

nameable

### Base Classes

SOMObject

### Metaclass

SOMClass

### Ancestor Classes

SOMObject

### New Methods

somrSetObjName  
somrGetObjName

### Overriding Methods

somlinit  
somUninit

## somrGetObjName Method

### Purpose

Returns a pointer to an object's name string.

### IDL Syntax

```
string somrGetObjName ( );
```

### Description

The **somrGetObjName** method returns a pointer to an object's name string.

Ownership of the string remains with the object.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRNameable</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

A pointer to a string (which could be NULL if the object was not assigned a name).

### Examples

```
if ( !strcmp( _somrGetObjName( anObject, Env), "Lassie" ) ) {  
    _bark( anObject ); }
```

### Original Class

**SOMRNameable**

### Related Information

Methods: **somrSetObjName**

SOMRNameable class

## somrSetObjName Method

### Purpose

Sets the name of a nameable object.

### IDL Syntax

```
void somrSetObjName (  
    in string name);
```

### Description

The **somrSetObjName** method sets the internal pointer to *name*.

This method transfers ownership of the string to the receiving object.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRNameable</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>name</i>	A pointer to the name of the object.

### Return Values

None.

### Examples

```
_somrSetObjName( anObject, Env, "Lassie" );
```

### Original Class

SOMRNameable

### Related Information

Methods: somrGetObjName



---

## SOMRReplicbl Class

### Description

The **SOMRReplicbl** class provides a link to the replica management subsystem. Any class derived from this class can have groups of instances that are replicas of each other. That is, the group of instances act as if they are one object. Changes to one replica are propagated to others in the group. All changes are applied in the same order to keep the replicas consistent.

To achieve replicability, the derived object must abide by the following rules [further derivations and contained (constituent) subobjects must abide by these rules as well]:

1. It must obtain a replica lock before updating its data and must release the same after the update. That is, the update methods must bracket their code with one of two possible method pairs:  
**somrLock** and **somrReleaseNPropagateUpdate** or  
**somrLockNLogOp** and **somrReleaseNPropagateOperation**
2. After obtaining the replica lock, if the object decides to abort an update operation, it must call the appropriate abort method:  
**somrReleaseLockNAbortUpdate** or  
**somrReleaseLockNAbortOp**
3. In case value logging is used, it must have an update language in which changes in the state of the object can be described.
4. In case value logging is used, it must provide a method to receive and interpret update messages propagated by other replicas. That is, it must implement the **somrApplyUpdates** method. When there are subobjects, this implementation should call them to interpret the updates appropriate to them.
5. It must have methods to get and set the complete state of the object (including any subobjects). That is, it must provide implementations for **somrGetState** and **somrSetState**.
6. It should be able to receive and interpret data replication directives (such as, LOST\_CONNECTION, BECOME\_STAND\_ALONE, and so forth).

### File Stem

replicbl

### Base Classes

SOMRNameable, SOMRLinearizable

### Metaclass

SOMClass

### Ancestor Classes

SOMRNameable, SOMRLinearizable, SOMObject

### New Methods

**somrApplyUpdates**  
**somrDoDirective**  
**somrGetSecurityPolicy**  
**somrLock**  
**somrLockNlogOp**

## **SOMRReplicbl class**

- somrPin**
- somrReleaseLockNAbortOp**
- somrReleaseLockNAbortUpdate**
- somrReleaseNPropagateUpdate**
- somrReleaseNPropagateOperation**
- somrReplnit**
- somrRepUninit**
- somrUnpin**

## **Overriding Methods**

- somlnit**
- somUninit**

## somrApplyUpdates Method

### Purpose

Interprets the buffer received as an update to its state.

### IDL Syntax

```
void somrApplyUpdates (
    string buffer,
    int bufferLen,
    int objIntId);
```

### Description

When doing value logging, the **somrApplyUpdates** method interprets the contents of the buffer received as an update to its state. The format of this update is exactly the same as the one used by the subclass implementor for the update buffer passed to the **somrReleaseNPropagateUpdate** method. (The length of the buffer is in the first **sizeof(long)** bytes of this string. Although typed as a string, the data area following the length field may contain NULLs.)

The **somrApplyUpdates** method is an obligation for a replicable object when value logging is being done. In this case, **somrApplyUpdates** *must* be overridden in a derived class.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>buffer</i>	A pointer to a character buffer representing update information.
<i>bufferLen</i>	The size of <i>buffer</i> .
<i>objIntId</i>	This parameter is reserved for future use.

### Return Values

None.

### Examples

If **somrGetState** is used to fill the buffer in **somrReleaseNPropagateUpdate**, then the following would be the implementation of **somrApplyUpdates**.

```
SOM_Scope void    SOMLINK somrApplyUpdates( AnObject somSelf,
                                           Environment *env, string buf,
                                           int len, int objIntId)
{
    /* parse the byte string in buf and apply the state
       changes to the object. */
    . . .
}
```

### Original Class

**SOMRReplicbl**

### Related Information

Methods: **somrReleaseNPropagateUpdate**, **somrSetState**, **somrGetState**

## somrDoDirective Method

### Purpose

Interprets a directive sent to a replica.

### IDL Syntax

```
void somrDoDirective (  
    in string str);
```

### Description

A directive is a message from the Replication Framework to a replica (actually to the application that is using the replica). A directive indicates that some condition has arisen asynchronously (not as a reaction to any request by the local replica). One handles directives by overriding the **somrDoDirective** method.

The **somrDoDirective** method is an obligation for replicable objects. It *must* be overridden in a derived class.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>str</i>	A string representing the directive.

### Return Values

None.

### Examples

Customize your method definition in the SOM-generated implementation file, where the ellipses represent application dependent code:

```
SOM_Scope void SOMLINK somrDoDirective( AnObject somSelf,  
                                         Environment *env, string str)  
{  
    AnObjectData *somThis = AnObjectGetData(somSelf);  
    if ( !strcmp( directive, "BECOME_STAND_ALONE" ) ) {  
        ... }  
    else if ( !strcmp( directive, "CONNECTION_LOST" ) ) {  
        ... }  
    else if ( !strcmp( directive, "CONNECTION_REESTABLISHED" ) ) {  
        ... }  
}
```

### Original Class

SOMRReplicbl

### Related Information

Methods: **somrReplnit**, **somrRepUnit**

## somrGetSecurityPolicy Method

### Purpose

Returns the security policy for replicated objects.

### IDL Syntax

```
long somrGetSecurityPolicy ( );
```

### Description

The **somrGetSecurityPolicy** method returns the security policy for replicated objects that either are non-persistent or are persistent but haven't been created yet. The returned value is used as the *mode* parameter of *open* for creating the *.scf* file.

If the **somrGetSecurityPolicy** method is not overridden, the default is to open the *.scf* file with read and write permission, as follows. On AIX, the default security policy is

```
S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH
```

On OS/2 and Windows, the default security policy is

```
S_IREAD | S_IWRITE
```

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

The **somrGetSecurityPolicy** method returns an integer representing the security policy of the receiving object.

### Examples

For AIX, one might override the security policy as follows:

```
int somrGetSecurityPolicy (SOMRReplicbl receiver
                          Environment *env);
{
    return S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH ;
}
```

### Original Class

**SOMRReplicbl**

## somrLock Method

### Purpose

Gets a lock on the replica of the object when doing value logging.

### IDL Syntax

```
void somrLock ( );
```

### Description

The **somrLock** method gets a lock on the current replica of the object. This method is used only when the replicated object is initialized for value logging (see **somrReplnit**). The exception raised indicates whether the lock was successfully obtained.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

None. The **somrLock** method can raise the following exceptions: SOMR\_DENIED, and SOMR\_TRYLATER.

### Examples

The code template for a method that modifies the value of a ReplicatedDog might look as follows:

```
dogMethod( ReplicatedDog somSelf, <parameters> ) {
    char *buf;
    Environment *Env = SOM_CreateLocalEnvironment();
    _somrLock( somSelf, Env );
    if (Env->_major == NO_EXCEPTION) {
        parent_dogMethod( somSelf, <parameters> )
        buf = <some algorithm to capture the change
              in the state of the object>;
        _somrReleaseNPropagateUpdate( somSelf, Env
                                     "ReplicatedDog",
                                     buf,
                                     <buf length in bytes>,
                                     0 );
    }
    else {
        /* code to handle failure to obtain a lock */
        switch (somriGetErrorCode(Env)){
        case SOMR_MASTERUNREACHABLE: ...
        case SOMR_UNAUTHORIZED: ...
        case SOMR_TIMEOUT: ...
        case SOMR_TRYLATER ...
        default: ...
        }
    }
}
```

### Original Class

SOMRReplicbl

### Related Information

Methods: **somrPin**, **somrReleaseLockNAbortUpdate**, **somrReleaseNPropagateUpdate**

## somrLockNlogOp Method

### Purpose

Gets a lock on the replica of the object and logs the method.

### IDL Syntax

```
void somrLockNlogOp (
    in string classname,
    in string methodname,
    in va_list *ap);
```

### Description

The **somrLockNlogOp** method gets a lock on the current replica of the object. This method is used only when the replicated object is initialized for operation logging (see **somrReplnit**). This method is the same as **somrLock**, but has the additional responsibility of logging the method that is requesting the lock.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>classname</i>	The name of the class of the object.
<i>methodname</i>	The name of the method to be logged.
<i>ap</i>	A pointer to a <b>va_list</b> that specifies the arguments with which <i>methodName</i> is called. If the callstyle of the method is IDL, then the first argument must be a pointer to the Environment structure.

### Return Values

None. The **somrLockNlogOp** method can raise the exceptions SOMR\_DENIED, and SOMR\_TRYLATER.

### Examples

```
#include <somrerrd.h>

dogMethod( ReplicatedDog somSelf, <parameters> ) {
    ReplicatedDogData *somThis = ReplicatedDogGetData(somSelf );
    Environment *Env = SOM_CreateLocalEnvironment();
    _somrLockNlogOp( somSelf,
                    Env,
                    "ReplicatedDog",
                    "dogMethod",
                    <parameters> );
    if (Env->_major == NO_EXCEPTION ) {
        parent_dogMethod( somSelf, <parameters> );
        ... <any other additional code> ...
        _somrReleaseNPropagateOperation( somSelf, Env );
    }
    else {
        /* code to handle failure to obtain a lock */
    }
}
```

### Original Class

SOMRReplicbl

### Related Information

Methods: **somrPin**, **somrReleaseLockNAbortOp**, **somrReleaseNPropagateOperation**

## somrPin Method

### Purpose

Pins the lock to this replica until **somrUnPin** is called.

### IDL Syntax

```
void somrPin ( );
```

### Description

The lock obtained by this replica stays with it until a call to **somrUnPin** is made. That is, it makes the replica lock un-preemptible.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

None. If the lock is denied, the exception SOMR\_DENIED is raised.

### Examples

Below is a projection of a sequence of requests to the Replication Framework that are spread over two update methods of a replicated object. Because of the pinning of the lock, the user is assured that the lock will not be lost between the two methods.

```
first method { somrLock( somself, ev );  
              somrPin( somSelf, ev );  
              ...  
              somrReleaseNPropagateUpdate( somself, ev, clsnm, buf, len , objId );  
  
second method { somrLock( somself, ev );  
               ...  
               somrReleaseNPropagateUpdate( somself, ev, clsnm, buf, len, objId );  
               somrUnPin( somself, ev );
```

### Original Class

SOMRReplicbl

### Related Information

Methods: **somrLock**, **somrLockNlogOp**, **somrUnPin**



## somrReleaseLockNAbortOp Method

### Purpose

Aborts the operation begun by calling the **somrLockNLogOp** method.

### IDL Syntax

```
void somrReleaseLockNAbortOp ( );
```

### Description

With operation logging, once a lock is obtained, either the **somrReleaseLockNAbortOp** method or the **somrReleaseNPropagateOperation** method must be called.

The **somrReleaseLockNAbortOp** method informs the Replication Framework that the user decided to abort an operation begun by calling the **somrLockNLogOp** method.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

None.

### Examples

```
#include <somrerrd.h>

dogMethod( ReplicatedDog somSelf, <parameters> ) {
  ReplicatedDogData *somThis = ReplicatedDogGetData(somSelf );
  Environment *Env = SOM_CreateLocalEnvironment();
  _somrLockNlogOp( somSelf,
                  Env,
                  "ReplicatedDog",
                  "dogMethod",
                  <parameters> );
  if (Env->_major == NO_EXCEPTION ) {
    parent_dogMethod( somSelf, <parameters> );
    ... <User now decides to abort> ...
    _somrReleaseLockNAbortOp( somSelf, Env );
  }
  else {
    /* code to handle failure to obtain a lock */
  }
}
```

### Original Class

SOMRReplicbl

### Related Information

Methods: **somrReleaseNPropagateOperation**, **somrLockNLogOp**

## somrReleaseLockNAbortUpdate Method

### Purpose

Aborts the operation begun by calling **somrLock**.

### IDL Syntax

```
void somrReleaseLockNAbortUpdate ( );
```

### Description

When doing value logging, once a lock is obtained, one of the following two methods must be called: either **somrReleaseLockNAbortUpdate** or **somrReleaseNPropagateUpdate**.

The **somrReleaseLockNAbortUpdate** method lets the Replication Framework know that the user decided to abort the operation begun by calling **somrLock**.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

None.

### Examples

```
dogMethod( ReplicatedDog somSelf, <parameters> ) {
    char *buf;
    Environment *Env = SOM_CreateLocalEnvironment();
    _somrLock( somSelf, Env );
    if (Env->_major == NO_EXCEPTION) {
        parent_dogMethod( somSelf, <parameters> )
        ...
        /* User now decides to abort */
        _somrReleaseLockNAbortUpdate( somSelf, Env,
                                     "ReplicatedDog",
                                     buf,
                                     <buf length in bytes>,
                                     0 );
    }
    else {
        /* code to handle failure to obtain a lock */
    }
}
```

### Original Class

**SOMRReplicbl**

### Related Information

Methods: **somrReleaseNPropagateUpdate**, **somrLock**

## somrReleaseNPropagateOperation Method

### Purpose

Releases the lock and propagates the operation log.

### IDL Syntax

```
void somrReleaseNPropagateOperation ( );
```

### Description

When doing operation logging, the **somrReleaseNPropagateOperation** method request the release of the lock and propagates the log of update operations to the other replicas.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

None.

### Examples

```
#include <somrerrd.h>

dogMethod( ReplicatedDog somSelf, <parameters> ) {
  ReplicatedDogData *somThis = ReplicatedDogGetData(somSelf );
  Environment *Env = SOM_CreateLocalEnvironment();
  _somrLockNlogOp( somSelf,
                  Env,
                  "ReplicatedDog",
                  "dogMethod",
                  <parameters> );
  if (Env->_major == NO_EXCEPTION ) {
    parent_dogMethod( somSelf, <parameters> );
    ... <any other additional code> ...
    _somrReleaseNPropagateOperation( somSelf, Env );
  }
  else {
    /* code to handle failure to obtain a lock */
  }
}
```

### Original Class

SOMRReplicbl

### Related Information

Methods: **somrReleaseLockNAbortOp**, **somrLockNlogOp**

## somrReleaseNPropagateUpdate Method

### Purpose

Requests the release of the lock and propagates the value of the replica.

### IDL Syntax

```
void somrReleaseNPropagateUpdate (
    in string clsname,
    in string buffer,
    in int bufferlen,
    in int intObjId);
```

### Description

When doing value logging, the **somrReleaseNPropagateUpdate** method calls the local replica manager to release a lock locally and to propagate local updates to the master and/or other shadows. This propagates the “value log” of state changes.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>clsname</i>	A character string representing the name of the class to be logged.
<i>buffer</i>	A character buffer for logged information. (The length of the string is in the first <b>sizeof(long)</b> bytes of this string. Although typed as a string, the data area following the length field may contain NULLs.)
<i>bufferlen</i>	An integer representing the size of the buffer required.
<i>intObjId</i>	This parameter is reserved for future use; it should always be set to 0.

### Return Values

None.

### Examples

The code template for a method that modifies the value of a ReplicatedDog might look as follows:

```
dogMethod( ReplicatedDog somSelf, <parameters> ) {
    char *buf;
    Environment *Env = SOM_CreateLocalEnvironment();
    _somrLock( somSelf, Env );
    if (Env->_major == NO_EXCEPTION) {
        parent_dogMethod( somSelf, <parameters> )
        buf = <some algorithm to capture the change
              in the state of the object>;
        __somrReleaseNPropagateUpdate( somSelf, Env,
                                       "ReplicatedDog",
                                       buf,
                                       <buf length in bytes>,
                                       0 );
    }
    else {
        /* code to handle failure to obtain a lock */
    }
}
```

**Original Class**

SOMRReplicbl

**Related Information**

Methods: somrReleaseLockNAbortUpdate, somrLock

## somrReplnit Method

### Purpose

Makes the object ready for replication.

### IDL Syntax

```
long somrReplnit (
    in char IType,
    in char mode);
```

### Description

The **somrReplnit** method prepares the object for replication. A derived object *must* call this method for activating replica control.

The *IType* parameter indicates the type of logging used. If *IType* is **v**, then value logging is used. If *IType* is **o**, then operation logging is used.

The parameter *mode* indicates whether the object is opened for reading (**r**) or writing (**w**).

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.
<i>IType</i>	A <b>char</b> indicating the type of logging used: <b>v</b> indicates <i>value logging</i> ; <b>o</b> indicates <i>operation logging</i> .
<i>mode</i>	A <b>char</b> indicating whether the object is open for reading ( <b>r</b> ) or writing ( <b>w</b> ).

### Return Values

The **somrReplnit** method returns 1 to indicate that this is the first replica to be activated (the master), or 0 indicates it is a shadow. If an error occurs, one of the following exceptions can be raised: SOMR\_MASTER\_UNREACHABLE and SOMR\_UNAUTHORIZED. It is also possible to obtain SOMR\_TRYLATER.

### Examples

```
#include <somrerrd.h>

Environment *Env;
int rc;
ReplicatedDog dog = ReplicatedDogNew();
Env = SOM_CreateLocalEnvironment();

    _somrSetObjName ( dog, Env, "Lassie" );
    rc = _somrReplnit( dog, Env, 'o', 'w');
    if (Env->_major == NO_EXCEPTION) {
        somPrintf(
            "Successfully initialized for replication. rc = %d\n",
                rc);
        ...
    }
    else {
        somPrintf("Initialization for replication failed\n");
        switch(somriGetErrorCode(Env)) {
            case SOMR_MASTERUNREACHABLE: ...
            case SOMR_UNAUTHORIZED: ...
            case SOMR_TRYLATER: ...
            default: ...
        }
    }
```

**Original Class**

SOMRReplicbl

**Related Information**

Methods: somrRepUnit

SOMRReplicbl class

## somrRepUninit Method

### Purpose

Destroys the setup for replication.

### IDL Syntax

```
void somrRepUninit ();
```

### Description

The **somrRepUninit** method destroys the setup for replication.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

None.

### Examples

```
_somrRepUninit (somSelf, Env);
```

### Original Class

**SOMRReplicbl**

### Related Information

Methods: **somrReplnit**



## somrUnPin Method

### Purpose

Unpins the lock so that it can be obtained by another replica.

### IDL Syntax

```
void somrUnPin ( );
```

### Description

The lock obtained by either **somrLock** or **somrLockNlogOp** can be pinned to the replica by **somrPin**. This means that the lock is not released until **somrUnPin** is executed by the pinning replica.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMRReplicbl</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the caller.

### Return Values

None.

### Examples

The **somrUnPin** and **somrReleaseNPropagateOperation** methods can be done in either order.

```
somrReleaseNPropagateOperation( somSelf, Env );
somrUnPin( somSelf, Env );
```

is equivalent to

```
somrUnPin( somSelf, Env );
somrReleaseNPropagateOperation( somSelf, Env );
```

Below is a projection of a sequence of requests to the Replication Framework that are spread over two update methods of a replicated object. Because of the pinning of the lock, the user is assured that the lock will not be lost between the two methods.

```
first method { somrLock( somself, ev );
               somrPin( somSelf, ev );
               ...
               somrReleaseNPropagateUpdate( somself, ev, clsnm, buf, len, objId );

second method { somrLock( somself, ev );
                ...
                somrReleaseNPropagateUpdate( somself, ev, clsnm, buf, len, objId );
                somrUnPin( somself, ev );
```

### Original Class

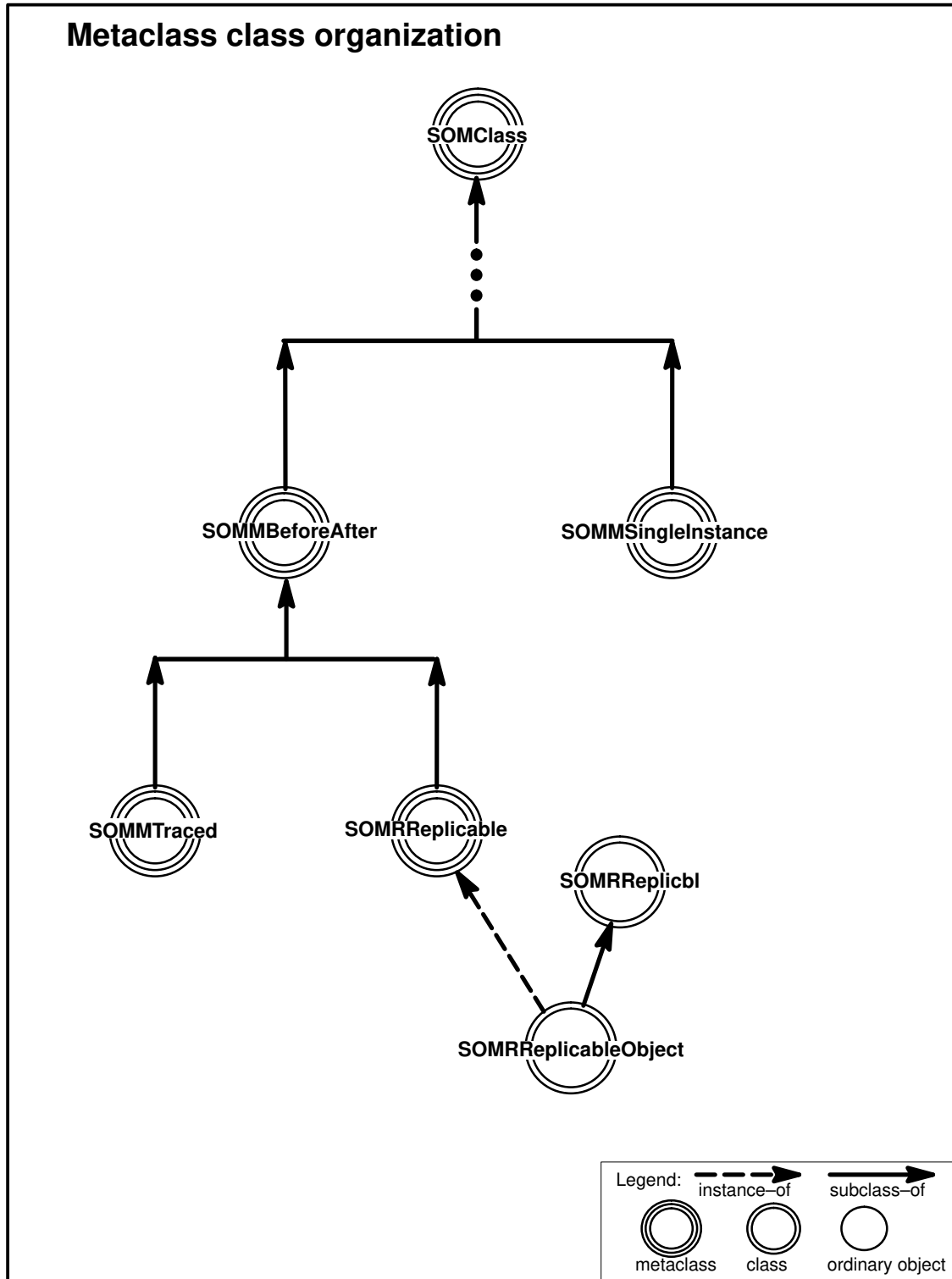
SOMRReplicbl

### Related Information

Methods: **somrPin**, **somrLock**, **somrLockNlogOp**



# Metaclass Framework Reference



---

## SOMMBeforeAfter Metaclass

### Description

**SOMMBeforeAfter** is a metaclass that defines two methods (**sommBeforeMethod** and **sommAfterMethod**) which are invoked before and after each invocation of every instance method. **SOMMBeforeAfter** is designed to be subclassed. Within the subclass, each of the two methods should be overridden with a method procedure appropriate to the particular application. The before and after methods are invoked on instances (ordinary objects) of a class whose metaclass is the subclass (or child) of **SOMMBeforeAfter**, whenever any method (*inherited* or *introduced*) of the class is invoked.

Caution: The **somDefaultInit** and **somFree** methods are among the methods that get before/after behavior. This implies that the following two obligations are imposed on the programmer of a **SOMMBeforeAfter** class. First, your implementation must guard against calling the **sommBeforeMethod** before **somDefaultInit** has executed, when the object is not yet fully initialized. Second, the implementation must guard against calling **sommAfterMethod** after **somFree**, at which time the object no longer exists.

**SOMMBeforeAfter** is thread-safe.

### File Stem

sombacIs

### New Methods

None.

### Overriding Methods

somDefaultInit

## sommAfterMethod Method

### Purpose

Specifies a method that is automatically called after execution of each client method.

### IDL Syntax

```
void sommAfterMethod (
    in SOMObject object,
    in somId methodId,
    in void *returnedvalue,
    in va_list ap);
```

### Description

The **sommAfterMethod** specifies a method that is automatically called after execution of each client method. The **sommAfterMethod** method is introduced in the **SOMMBeforeAfter** metaclass. The default implementation does nothing until it is overridden. The **sommAfterMethod** method is not called directly by the user. To define the desired “after” method, **sommAfterMethod** must be overridden in a metaclass that is a subclass (child) of the **SOMMBeforeAfter** metaclass.

Caution: **somFree** is among the methods that get before/after behavior, which implies that the following obligation is imposed on the programmer of a **sommAfterMethod**. Specifically, care must be taken to guard against **sommAfterMethod** being called after **somFree**, at which time the object no longer exists.

### Parameters

Refer to the Example’s diagram for further clarification of these arguments.

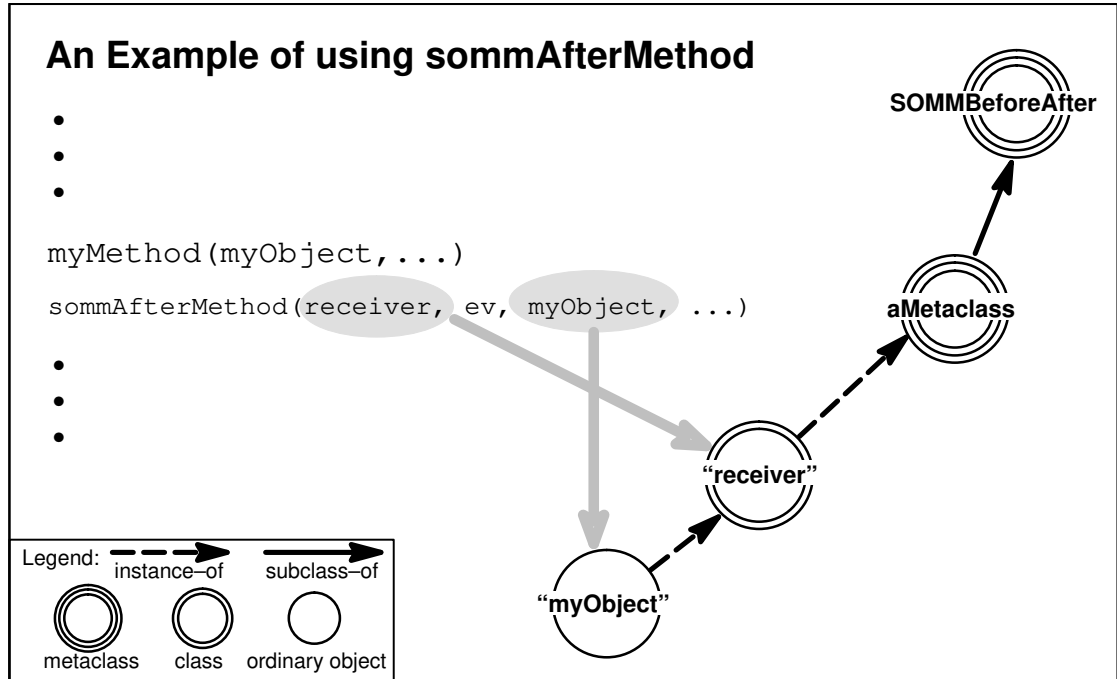
<i>receiver</i>	A pointer to an object (class) of metaclass <b>SOMMBeforeAfter</b> representing the class object that supports the method (such as, “myMethod”) for which the “after” method will apply.
<i>ev</i>	A pointer where the method can return exception information if an error is encountered. The dispatch method of <b>SOMMBeforeAfter</b> sets this parameter to NULL before dispatching the first <b>sommBeforeMethod</b> .
<i>object</i>	A pointer to the instance of the receiver on which the method is invoked.
<i>methodId</i>	The SOM ID of the method (such as, “myMethod”) that was invoked.
<i>returnedvalue</i>	A pointer to the value returned by invoking the method (“myMethod”) on an object.
<i>ap</i>	The list of input arguments to the method (“myMethod”).

### Return Value

None.

### Example

The following figure shows an invocation of “myMethod” on “myObject”. Because “myObject” is an instance of a class whose metaclass is a subclass of **SOMMBeforeAfter**, “myMethod” is followed by an invocation of **sommAfterMethod** (which is shown in smaller type to denote that the user does not actually code the method). The adjacent figure illustrates the meaning of the parameters to **sommAfterMethod**.



### Original Class

SOMMBeforeAfter

### Related Information

Methods: sommBeforeMethod

## sommBeforeMethod Method

### Purpose

Specifies a method that is automatically called before execution of each client method.

### IDL Syntax

```
boolean sommBeforeMethod (
    in SOMObject object,
    in somId methodId,
    in va_list ap);
```

### Description

The **sommBeforeMethod** specifies a method that is automatically called before execution of each client method. The **sommBeforeMethod** method is not called directly by the user. To define the desired “before” method, **sommBeforeMethod** must be overridden in a metaclass that is a subclass (child) of **SOMMBeforeAfter**. The default implementation does nothing until it is overridden.

Caution: **somDefaultInit** is among the methods that get before/after behavior, which implies that the following obligation is imposed on the programmer of a **sommBeforeMethod**. Specifically, care must be taken to guard against **sommBeforeMethod** being called before the **somDefaultInit** method has executed and the object is not yet fully initialized.

### Parameters

Refer to the Example’s diagram for further clarification of these arguments.

<i>receiver</i>	A pointer to an object (class) of metaclass <b>SOMMBeforeAfter</b> representing the class object that supports the method (such as, “myMethod”) for which the “before” method will apply.
<i>ev</i>	A pointer where the method can return exception information if an error is encountered. The dispatch method of <b>SOMMBeforeAfter</b> sets this parameter to NULL before dispatching the first <b>sommBeforeMethod</b> .
<i>object</i>	A pointer to the instance of the <i>receiver</i> on which the method is invoked.
<i>methodId</i>	The SOM ID of the method (such as, “myMethod”) that was invoked.
<i>ap</i>	The list of input arguments to the method (“myMethod”).

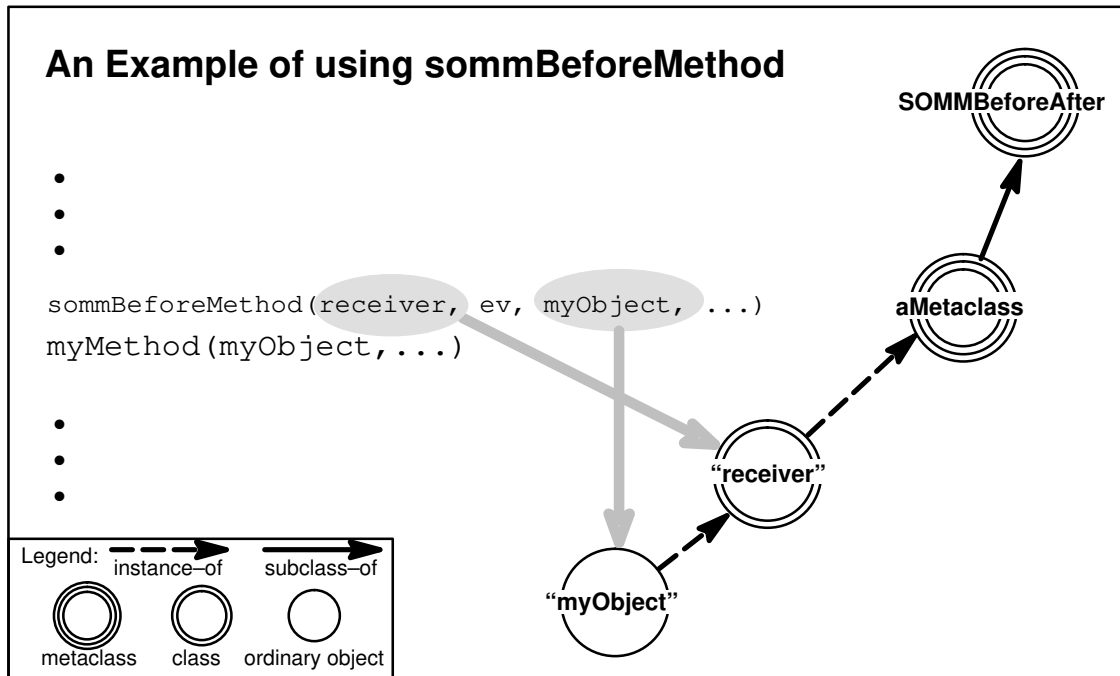
### Return Value

A **boolean** that indicates whether or not before/after dispatching should continue. If the value is TRUE, normal before/after dispatching continues. If the value is FALSE, the dispatching skips to the **sommAfterMethod** associated with the preceding **sommBeforeMethod**. This implies that the **sommBeforeMethod** must do any post-processing that might otherwise be done by the **sommAfterMethod**. Because before/after methods are paired within a **SOMMBeforeAfter** metaclass, this design eliminates the complexity of communicating to the **sommAfterMethod** that the **sommBeforeMethod** returned FALSE.

### Example

The following figure shows an invocation of “myMethod” on “myObject”. Because “myObject” is an instance of a class whose metaclass is a subclass of **SOMMBeforeAfter**, “myMethod” is preceded by an invocation of **sommBeforeMethod** (which is shown in smaller type to denote that the user does not actually code the method). The adjacent figure illustrates the meaning of the parameters to **sommBeforeMethod**

## SOMMBeforeAfter metaclass



## Original Class

**SOMMBeforeAfter**

## Related Information

**Methods:** `sommAfterMethod`



---

## SOMMSingleInstance Metaclass

### Description

**SOMMSingleInstance** can be specified as the metaclass when a class implementor is defining a class for which only one instance can ever be created. The first call to `<className>New` in C, the **new** operator in C++, or the **somNew** method creates the one possible instance of the class. Thereafter, any subsequent “new” calls return the first (and only) instance.

Alternatively, the *method* **sommGetSingleInstance** can be used to accomplish the same purpose. The method offers an advantage in that the call site explicitly shows that something special is occurring and that a new object is not necessarily being created.

**SOMMSingleInstance** is thread-safe.

### File Stem

snglicls

### Base Class

SOMClass

### Metaclass

SOMClass

### Ancestor Classes

SOMClass, SOMObject

### New Methods

sommGetSingleInstance

### Overriding Methods

somlnit  
somNew

## sommGetSingleInstance Method

### Purpose

Gets the one instance of a specified class for which only a single instance can exist.

### IDL Syntax

```
SOMObject sommGetSingleInstance ( );
```

### Description

The **sommGetSingleInstance** method gets a pointer to the one instance of a class for which only a single instance can exist. A class can have only a single instance when its metaclass is the **SOMMSingleInstance** metaclass (or is a subclass of it).

The first call to `<className>New` in C, the `new` operator in C++, or the **somNew** method creates the one possible instance of the class. Thereafter, any subsequent “new” calls return the first (and only) instance. Using the **sommGetSingleInstance** method offers an advantage, however, in that the call site explicitly shows that something special is occurring and that a new object is not necessarily being created. (That is, the **sommGetSingleInstance** method creates the single instance if it does not already exist.)

### Parameters

<i>receiver</i>	A pointer to a class object whose metaclass is <b>SOMMSingleInstance</b> (or is a subclass of it).
<i>env</i>	A pointer where the method can return exception information if an error is encountered.

### Return Value

The **sommGetSingleInstance** method returns a pointer to the single instance of the specified class.

### Example

Suppose the class “XXX” is an instance of **SOMMSingleInstance**; then the following C code fragment passes the assertions.

```
x1 = XXXNew();
x2 = XXXNew();
assert( x1 == x2 );
x3 = _sommGetSingleInstance( _somGetClass( x1 ), env );
assert( x2 == x3 );
```

Note that the method **sommGetSingleInstance** is invoked on the class object, because **sommGetSingleInstance** is a method introduced by the metaclass **SOMMSingleInstance**.

### Original Class

**SOMMSingleInstance**

---

## SOMMTraced Metaclass

### Description

**SOMMTraced** is a metaclass that facilitates tracing of method invocations. Whenever a method (inherited or introduced) is invoked on an instance (simple object) of a class whose metaclass is **SOMMTraced**, a message prints to standard output giving the method parameters; then, after completion, a second message prints giving the returned value.

There is one more step for using **SOMMTraced**: nothing prints unless the environment variable `SOMM_TRACED` is set. If it is set to the empty string, *all* traced classes print. If the environment variable `SOMM_TRACED` is not the empty string, it should be set to the list of names of classes that should be traced. For example, for *csH* users, the following command turns on printing of the trace for “Collie” and “Chihuahua”, but not for any other traced class:

```
setenv SOMM_TRACED "Collie Chihuahua"
```

**SOMMTraced** is thread-safe.

### File Stem

`somtrcls`

### Base Class

`SOMMBeforeAfter`

### Ancestor Classes

`SOMMBeforeAfter`, `SOMClass`, `SOMObject`

### Attributes

**boolean `sommTracelsOn`**

This attribute indicates whether or not tracing is turned on for a class.  
This gives dynamic control over the trace facility.

### New Methods

None.

### Overriding Methods

`sommBeforeMethod`  
`sommAfterMethod`

---

## SOMRReplicable Metaclass

### Description

**SOMRReplicable** is the metaclass for **SOMRReplicableObject**. These two illustrate a second way to use a metaclass to impart properties. Here, the metaclass is not intended to have any other instance than **SOMRReplicableObject**, which imparts the desired property to ordinary objects.

The need for this combination arises from a requirement for replicable classes to support the **somrReplicableExemptMethod** method. Although this method could be introduced by **SOMRReplicable**, each class must override it. To override a method in IDL, however, the method must be introduced by a parent. Thus, the best design (with respect to usability) has **somrReplicableExemptMethod** introduced by **SOMRReplicableObject** so that the method is easily overridden.

### File Stem

somrmcls

### Base Class

SOMMBeforeAfter

### Ancestor Classes

SOMMBeforeAfter, . . ., SOMClass

### New Methods

None.

### Overriding Methods

sommBeforeMethod, sommAfterMethod

---

## SOMRReplicableObject Class

### Description

This base class makes the Replication Framework easy to use. The only obligations of an application programmer are to:

Override **somrDoDirective**, **somrGetState**, and **somrSetState**, and

Invoke **somrSetObjName** and **somrReplnit** on instances of **SOMRReplicableObject**.

### File Stem

somrcls

### Base Class

SOMRReplicbl

### Metaclass

SOMRReplicable

### Ancestor Classes

SOMRReplicbl, SOMRNameable, SOMRLinearizable, SOMObject

### New Methods

somrLoggingType

somrReplicableExemptMethod

### Overriding Methods

somrReplnit

somrApplyUpdates

## somrLoggingType Method

### Purpose

Enables querying of the logging type for a replicable object.

### IDL Syntax

```
char somrLoggingType ( );
```

### Description

The **somrLoggingType** method allows one to query a replicable object for its logging type. The method is used by the overrides of **sommBeforeMethod** and **sommAfterMethod** in the **SOMRReplicable** metaclass.

### Parameters

<i>receiver</i>	A pointer to an object that is a <b>SOMRReplicableObject</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

Either 'o' or 'v' depending on the logging type that was set by **somrReplnit**. (If **somrReplnit** has not been invoked, the result is unspecified.)

### Example

```
RepObject x;  
x = RepObjectNew();  
_somrSetObjName(x, ev, "aRepObject");  
_somrReplnit(x, ev, 'o', 'w');  
if ( 'o' == _somrLoggingType(x, ev) )  
    somPrintf( "This will print" );
```

### Original Class

**SOMRReplicableObject**

### Related Information

Methods: **somrReplnit**

## somrReplicableExemptMethod Method

### Purpose

Indicates which methods are exempt from the before/after methods in **SOMRReplicable**.

### IDL Syntax

```
boolean somrReplicableExemptMethod (in somId methodId);
```

### Description

Methods that do not update a replicated object need not have their effects propagated to all replicas (as a matter of fact, it is quite inefficient). One can indicate which methods of a replicable class are read-only by having **somrReplicableExemptMethod** return TRUE for those methods.

Note that methods supported by **SOMRReplicableObject** are automatically exempted. This includes all methods introduced by **SOMObject**, **SOMRReplicbl**, **SOMRNameable**, and **SOMRLinearizable**.

### Parameters

<i>receiver</i>	A pointer to an object that is a <b>SOMRReplicableObject</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>methodId</i>	The SOM ID of the method that was invoked.

### Return Value

Returns TRUE if the **sommBeforeMethod** and **sommAfterMethod** (in **SOMRReplicable**) should do nothing for the *methodId*. If FALSE is returned, the effect of the method is propagated to all replicas.

### Example

By overriding **somrReplicableExemptMethod** as follows, you can assure that the effect of the method named "foo" is not propagated to replicas.

```
boolean somrReplicableExemptMethod( somId methodId ) {
    if ( somCompareIds( methodId, somIdFromString("foo") ) )
        return TRUE;
    else
        return FALSE;
}
```

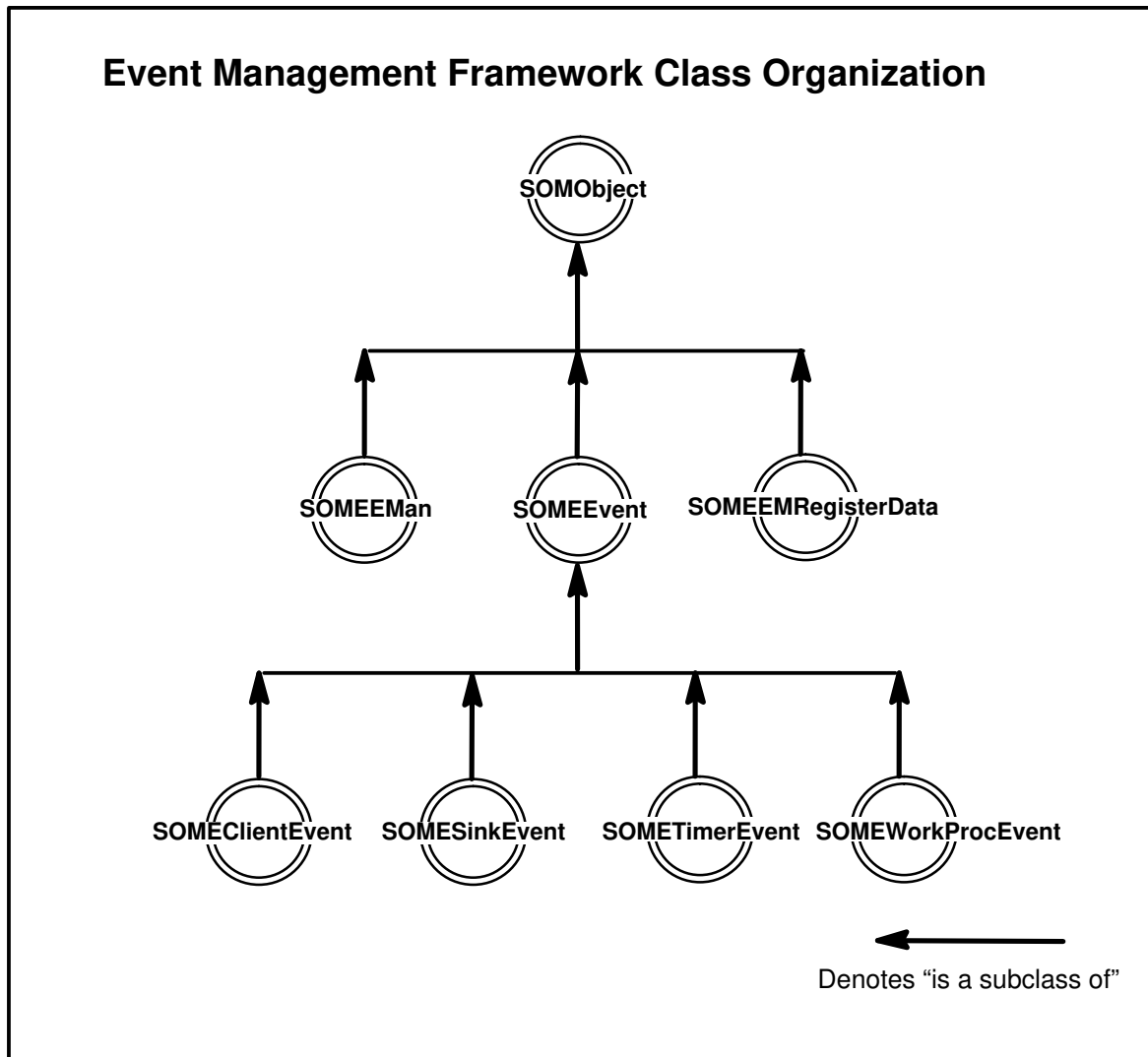
### Original Class

**SOMRReplicableObject**





# Event Management Framework Reference



---

## SOMEClientEvent Class

### Description

This class describes generic client events within the Event Manager. Client Events are defined, created, processed and destroyed entirely by the application. The application can queue several types of client events with EMan. When a client event occurs, EMan passes an instance of this class to the callback routine. The callback can query this object about its type and obtain any event-specific information.

### File Stem

clientev

### Base

SOMEEvent

### Metaclass

SOMClass

### Ancestor Classes

SOMEEvent SOMObject

### New Methods

somevGetEventClientData  
somevGetEventClientType  
somevSetEventClientData  
somevSetEventClientType

### Overriding Methods

somlnit

## somevGetEventClientData Method

### Purpose

Returns the user-defined data associated with a client event.

### IDL Syntax

```
void* somevGetEventClientData ( );
```

### Description

This method returns the user-defined data (if any) associated with the Client Event object. This associated data for a given client event type is passed to EMan at the time of registration.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEClientEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

A pointer to user-defined client event data.

### Original Class

**SOMEClientEvent**

### Related Information

Methods: **somevSetEventClientData**

SOMEClientEvent class

## somevGetEventClientType Method

### Purpose

Returns the type name of a client event.

### IDL Syntax

```
string somevGetEventClientType ( );
```

### Description

This method returns the client event type of the Client Event object. Client event type is a string name assigned to the event by the application at the time of registering the event.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEClientEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

A null terminated string identifying the client event type.

### Original Class

**SOMEClientEvent**

### Related Information

Methods: **somevSetEventClientType**

## somevSetEventClientData Method

### Purpose

Sets the user-defined data of a client event.

### IDL Syntax

```
void somevSetEventClientData (  
                                in void* clientData);
```

### Description

This method sets the user-defined event data (if any) of the Client Event object. This associated data for a given client event type is passed to EMan at the time of registration.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEClientEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>clientData</i>	A pointer to user-defined data for this client event.

### Return Value

None.

### Original Class

**SOMEClientEvent**

### Related Information

Methods: **somevGetEventClientData**

SOMEClientEvent class

## somevSetEventClientType Method

### Purpose

Sets the type name of a client event.

### IDL Syntax

```
void somevSetEventClientType (  
    in string clientType);
```

### Description

This method sets the client event type field of the Client Event object. Client event type is a string name assigned to the event by the application at the time of registering the event.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEClientEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>clientType</i>	A null terminated character string identifying the client event type. The contents of this string are entirely up to the user. However, while using class libraries that also use client events one must make sure that there are no name collisions.

### Return Value

None.

### Original Class

**SOMEClientEvent**

### Related Information

Methods: **somevGetEventClientType**

---

## SOMEEMan Class

### Description

The Event Manager class (EMan for short) is used to handle several input events. The main purpose of this class is to provide a service that can do a blocked (or timed) wait on several event sources concurrently. Typically, in a main program, one registers an interest in an event type with EMan and specifies a callback (a procedure or a method) to be invoked when the event of interest occurs. After all the necessary registrations are complete, the main program ends with a call to **someProcessEvent** in EMan. This call is non-returning. EMan then waits on all registered event sources. The application is completely event driven at this point (that is, it does something only when an event occurs). The control returns to EMan after processing each event. Further registrations can be done from within the callback routines. Unregistrations can also be done from within the callback routines.

For applications that want to have their own main loop, EMan provides a non-blocking call (the **someProcessEvent** method), which processes just one event (if any) and returns to the main loop immediately. Note that when this call is the only one in the application's main loop, CPU cycles are wasted in constantly polling for events. In this situation, the non-returning form of the **someProcessEvent** call is preferable.

#### AIX Specifics:

On AIX this event manager supports Timer, Sink (any file, pipe, socket, or Message Queue), Client and WorkProc events.

#### OS/2 and Windows Specifics:

On OS/2 and Windows, this event manager supports Timer, Sink (sockets only), Client, and WorkProc events.

#### Thread Safety:

To cope with multi-threaded applications on OS/2, the event-manager methods are mutually exclusive (that is, at any time only one thread can be executing inside of EMan). If an application thread needs to stop EMan from running (that is, to achieve mutual exclusion with EMan), it can use the two methods **someGetEManSem** and **someReleaseEManSem** to acquire and release EMan semaphore(s). On AIX or Windows, since threads are not supported (at present), calling these two methods has no effect.

### File Stem

eman

### Base Class

SOMObject

### Metaclass

SOMMSingleInstance

### Ancestor Classes

SOMObject

### New Methods

**someGetEManSem**  
**someReleaseEManSem**  
**someChangeRegData**  
**someProcessEvent**

## **SOMEEMan class**

- someProcessEvent**
- someQueueEvent**
- someRegister**
- someRegisterEv**
- someRegisterProc**
- someShutdown**
- someUnRegister**

## **Overriding Methods**

- somInit**
- somUninit**



## someChangeRegData Method

### Purpose

Changes the registration data associated with a specified registration ID.

### IDL Syntax

```
void someChangeRegData (
    in long registrationId,
    in SOMEEMRegisterData registerData);
```

### Description

This method is called to change the registration data associated with an existing registration of EMan. The existing registration is identified by the *registrationId* parameter. This ID must be the one returned by EMan when the event interest was originally registered with EMan. Further, the registration must be active (that is, it must not have been unregistered). The result of providing a non-existent or invalid registration ID is a “no op”.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>registrationId</i>	The registration ID of the event interest whose data is being changed.
<i>registerData</i>	A pointer to the registration data object whose contents will replace the existing registration information with EMan.

### Return Value

None.

### Example

```
#include <eman.h>
SOMEEMan *EManPtr;
SOMEEMRegisterData *data;
Environment *Ev;
long RegId;

...
_someChangeRegData (EManPtr, Ev, RegId, data);
```

### Original Class

**SOMEEMan**

### Related Information

Methods: **someRegister**, **someRegisterEv**, **someRegisterProc**

## someGetEManSem Method

### Purpose

Acquires EMan semaphore(s) to achieve mutual exclusion with EMan's activity.

### IDL Syntax

```
void someGetEManSem ( );
```

### Description

When EMan is used on OS/2, multiple threads can invoke methods on EMan concurrently. EMan protects its internal data by acquiring SOM toolkit semaphore(s). The same semaphore(s) are made available to users of EMan through the methods **someGetEManSem** and **someReleaseEManSem**. If an application desires to prevent EMan event processing from interfering with its own activity (in another thread, of course), then it can call the **someGetEManSem** method and acquire EMan semaphore(s). EMan activity will resume when the application thread releases the same semaphore(s) by calling **someReleaseEManSem**.

Callers should not hold this semaphore for too long, since it essentially stops EMan activity for that duration and may cause EMan to miss some important event processing. The maximum duration for which one can hold this semaphore depends on how frequently EMan must process events.

On AIX or Windows, calling this method has no effect.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

None.

### Example

```
#include <eman.h>
SOMEEMan *EManPtr;
Environment *Ev;

...
_someGetEManSem(EManPtr, Ev);
/* Do the work that needs mutual exclusion with EMan */
_someReleaseEManSem(EManPtr, Ev);
```

### Original Class

SOMEEMan

### Related Information

Methods: **someReleaseEManSem**

## someProcessEvent Method

### Purpose

Processes one event.

### IDL Syntax

```
void someProcessEvent (
    in unsigned long mask);
```

### Description

Processes one event. This call is non-blocking. If there are no events to process it returns immediately. The mask specifies which events to process. The mask is formed by OR'ing the bit constants specified in "eventmsk.h".

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>mask</i>	A bit mask indicating the types of events to look for and process.

### Return Value

None.

### Example

```
#include <eman.h>

main()
{
    Environment *testEnv = somGetGlobalEnvironment();
    SOMEEMan *some_gEMan = SOMEEManNew();
    /* Do some registrations */
    ...
    while (1) {
        _someProcessEvent (some_gEMan, testEnv,
                           EMProcessTimerEvent |
                           EMProcessSinkEvent |
                           EMProcessClientEvent );
        /*** Do other main loop work, if needed. ***/
    }
} /* end of main */
```

### Original Class

**SOMEEMan**

### Related Information

Methods: **someProcessEvents**, **someRegister**, **someRegisterProc**, **someRegisterEv**

## someProcessEvent Method

### Purpose

Processes infinite events.

### IDL Syntax

```
void someProcessEvent ( );
```

### Description

This call loops forever waiting for events and dispatching them. The only way this can be broken is by calling **someShutdown** in a callback routine. It is a programming error to call this method without having registered interest in any events with EMan. Typically, a call to this method is the last statement in an application's main program.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

None.

### Example

```
#include <eman.h>

main()
{
    Environment *testEnv = somGetGlobalEnvironment ();
    SOMEEMan *some_gEMan = SOMEEManNew ();
    /* Do some registrations */
    ...
    _someProcessEvent (some_gEMan, testEnv);
} /* end of main */
```

### Original Class

**SOMEEMan**

### Related Information

Methods: **someProcessEvent**, **someRegister**, **someRegisterProc**, **someRegisterEv**

## someQueueEvent Method

### Purpose

Enqueues the specified client event.

### IDL Syntax

```
void someQueueEvent (
    in SOMEClientEvent event);
```

### Description

Client events are defined, created, processed and destroyed by the application. EMan simply provides a means to enqueue and dequeue client events. Client events can be used in several ways. For example, if an application component wants to handle an input message arriving on a socket at a later time than when it arrives, it can receive the message in the socket callback routine, create a client event out of it, and queue it with EMan. EMan can be asked for the client event at a later time when the application is ready to handle it. Client events can also be useful to hide the origin of event sources (that is, the original event handlers receive the events and create client events in their place).

Dequeue is not a user-visible operation. Once a client event is queued, only EMan can dequeue it.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>event</i>	A pointer to the <b>SOMEClientEvent</b> object.

### Return Value

None.

### Example

```
#include <eman.h>
SOMEClientEvent *clientEvent1;

clientEvent1 = SOMEClientEventNew();
/* create a client event of type "ClientType1" */
_somevSetEventClientType( clientEvent1, testEnv, "ClientType1" );
_somevSetEventClientData( clientEvent1, testEnv, "Test Msg");
...

/* whenever it is desired to cause this client event to happen,
   call someQueueEvent Method with this clientEvent */
_someQueueEvent( some_gEMan, env, clientEvent1);
```

### Original Class

**SOMEEMan**

## someRegister Method

### Purpose

Registers an object/method pair with EMan, given a specified *registerData* object.

### IDL Syntax

```
long someRegister (
    in SOMEEMRegisterData registerData,
    in SOMObject targetObject,
    in string targetMethod,
    in void *targetData );
```

### Description

This method allows for registering an event of interest with EMan, with an object method as the callback. It is assumed that the target method has been declared as using OIDL callstyle. The event of interest and its details are filled in a registration data object *registerData*. The information about the callback routine is indicated by *targetObject* and *targetMethod*.

A mismatch between the target method's callstyle and the registration method used (that is, **someRegister** vs. **someRegisterEv**) can result in unpredictable results.

Note: The target method is called using name-lookup method resolution.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>registerData</i>	A pointer to the registration data object that contains all the necessary information about the event for which an interest is being registered with EMan.
<i>targetObject</i>	A pointer to the object that is the target of the callback method.
<i>targetMethod</i>	The name of the callback method.
<i>targetData</i>	A pointer to a data structure to be passed to the callback method when the event occurs.

### Return Value

The registration ID.

### Example

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

Environment *testEnv = somGetGlobalEnvironment();
some_gEMan = SOMEEManNew();          /* create an EMan object */
data = SOMEEMRegisterDataNew( );    /* create a reg data object */
target = EMObjectNew();             /* create a target object */

/* reRegister a timer event */
_someClearRegData( data, env );
_someSetRegDataEventMask( data, env, EMTimerEvent, NULL );
_someSetRegDataTimerInterval( data, env, 100 );
regId1 = _someRegister( some_gEMan, env, data, target,
                        "eventMethod", "Timer 100" );
```

## Original Class

SOMEEMan

## Related Information

**Methods:** `someRegisterEv`, `someRegisterProc`, `someUnRegister`

Also see the **callstyle** modifier of the SOM Interface Definition Language described in Chapter 4, “SOM IDL and the SOM Compiler” in the *SOM Toolkit User’s Guide*.

## someRegisterEv Method

### Purpose

Registers the (object, method, **Environment** parameter) combination of a callback with EMan, given a specified *registerData* object.

### IDL Syntax

```
long someRegisterEv (
    in SOMEEMRegisterData registerData,
    in SOMObject targetObject,
    inout Environment callbackEv,
    in string targetMethod,
    in void *targetData );
```

### Description

This method allows for registering an event interest with EMan with an object method as callback. The *callbackEv* is used as the environment pointer when EMan makes the callback. It is assumed that the target method has been declared as using IDL callstyle. The event of interest and its details are filled in a registration data object *registerData*. The information about the callback routine is indicated by *targetObject* and *targetMethod*.

A mismatch in the target method's callstyle and the registration method called (**someRegister** vs. **someRegisterEv**) can result in unpredictable results.

Note: The target method is called using name-lookup method resolution.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>registerData</i>	A pointer to registration data object that contains all the necessary information about the event for which an interest is being registered with EMan.
<i>targetObject</i>	A pointer to the object which is the target of the callback method
<i>callbackEv</i>	A pointer to the Environment structure to be passed to the callback method
<i>targetMethod</i>	The name of the callback method.
<i>targetData</i>	A pointer to a data structure to be passed to the callback method when the event occurs.

### Return Value

The registration ID.



## Example

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

Environment *testEnv = somGetGlobalEnvironment();
Environment *targetEv = somGetGlobalEnvironment();
some_gEMan = SOMEEManNew();          /* create an EMan object */
data = SOMEEMRegisterDataNew( );    /* create a reg data object */
target = EMOBJECTNew();             /* create a target object */

/* reRegister a timer event */
_someClearRegData( data, env );
_someSetRegDataEventMask( data, env, EMTimerEvent, NULL );
_someSetRegDataTimerInterval( data, env, 100 );
regIdl = _someRegisterEv( some_gEMan, env, data, target, targetEv,
                          "eventMethod", "Timer 100" );
/* eventMethod of target is assumed to use callstyle=idl */
```

## Original Class

SOMEEMan

## Related Information

**Methods:** `someRegister`, `someRegisterProc`, `someUnRegister`

Also see the **callstyle** modifier in the SOM Interface Definition Language described in Chapter 4, "SOM IDL and the SOM Compiler" in the *SOM Toolkit User's Guide*.

## someRegisterProc Method

### Purpose

Register the procedure with EMan given the specified *registerData*.

### IDL Syntax

```
long someRegisterProc (
    in SOMEEMRegisterData registerData,
    in EMRegProc *targetProcedure,
    in void *targetData );
```

### Description

The **someRegisterProc** method allows for registering an event of interest with EMan, with a specified procedure as the callback. The event of interest and its details are provided through a registration data object *registerData*. The information about the callback procedure is indicated by *targetProcedure*.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>registerData</i>	A pointer to registration data object that contains all the necessary information about the event for which an interest is being registered with EMan.
<i>targetProcedure</i>	A pointer to the procedure (callback) that is called when the registered event occurs.
<i>targetData</i>	A pointer to a data structure to be passed to the callback procedure when the event occurs.

### Return Value

The registration ID.

### Example

```
#include <eman.h>

void MyCallBack(SOMEEvent *event, void *somedata){
    ...
}

Environment *testEnv = somGetGlobalEnvironment();
some_gEMan = SOMEEManNew(); /* create an EMan object */
data = SOMEEMRegisterDataNew( ); /* create a reg data object */

/* reRegister a timer event */
_someClearRegData( data, env );
_someSetRegDataEventMask( data, env, EMTimerEvent, NULL );
_someSetRegDataTimerInterval( data, env, 100 );
regId1 = _someRegisterProc( some_gEMan, env, data,
                           MyCallBack, "Timer 100" );
```

### Original Class

SOMEEMan

### Related Information

Methods: **someRegister**, **someRegisterEv**, **someUnRegister**

## someReleaseEManSem Method

### Purpose

Releases the semaphore obtained by the **someGetEManSem** method.

### IDL Syntax

```
void someReleaseEManSem ( );
```

### Description

When EMan is used on OS/2, multiple threads can invoke methods on EMan concurrently. EMan protects its internal data by acquiring SOM toolkit semaphore(s). The same semaphore(s) are made available to users of EMan through the methods **someGetEManSem** and **someReleaseEManSem**. If an application desires to prevent EMan's event processing from interfering with its own activity (in another thread, of course), then it can call the **someGetEManSem** method and acquire EMan semaphore(s). EMan activity will resume when the application thread releases the same semaphore(s) by calling **someReleaseEManSem**.

Callers should not hold this semaphore for too long, since it essentially stops EMan activity for that duration and may cause EMan to miss some important event processing. The maximum duration for which one can hold this semaphore depends on how frequently EMan must process events.

On AIX or Windows, calling this method has no effect.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

None.

### Example

```
#include <eman.h>
SOMEEMan *EManPtr;
Environment *Ev;

...
_someGetEManSem(EManPtr, Ev);
/* Do the work that needs mutual exclusion with EMan */
_someReleaseEManSem(EManPtr, Ev);
```

### Original Class

SOMEEMan

### Related Information

Methods: **someGetEManSem**

## someShutdown Method

### Purpose

Shuts down an EMan event loop. (That is, this makes the **someProcessEvents** return!)

### IDL Syntax

```
void someShutdown ();
```

### Description

This can be called from a callback routine to break the someProcessEvents loop.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

None.

### Example

```
#include <eman.h>
SOMEEMan *some_gEMan;

void MyCallback(SOMEEvent *event, void *somedata){
    ...
    _someShutdown(some_gEMan, env);
}

main()
{
    Environment *testEnv = somGetGlobalEnvironment();
    SOMEEMan *some_gEMan = SOMEEManNew();
    /* Do some registrations. At least one involving MyCallback */
    ...
    _someProcessEvents(some_gEMan, testEnv);
}
```

### Original Class

**SOMEEMan**

### Related Information

Methods: **someProcessEvents**

## someUnRegister Method

### Purpose

Unregisters the event interest associated with a specified *registrationId* within EMan.

### IDL Syntax

```
void someUnRegister (
    in long registrationId);
```

### Description

When an application is no longer interested in a given event, it can unregister the event interest from EMan. EMan will stop making callbacks on this event, even if the event source continues to be active and generates events.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMan</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>registrationId</i>	The registration ID of the event that needs to be unregistered.

### Return Value

None.

### Example

```
#include <eman.h>
long regId1;

...
/* Register a timer */
regId1 = _someRegisterEv( some_gEMan, env, data, target, targetEv,
    "eventMethod", "Timer 100" );
....
/* Unregister the timer */
_someUnRegister(some_gEMan, env, regId1);
```

### Original Class

**SOMEEMan**

### Related Information

Methods: **someRegister**, **someRegisterEv**, **someRegisterProc**

---

## SOMEEMRegisterData Class

### Description

This class is used for holding registration information for event types to be registered with EMan. EMan extracts all needed information from this object and saves the information in its internal data structures. An instance of this class must be created, properly initialized, and passed to the registration methods of EMan for registering interest in any kind of event.

### File Stem

emregdat

### Base

SOMObject

### Metaclass

SOMClass

### Ancestor Classes

SOMObject

### New Methods

someClearRegData  
someSetRegDataClientType  
someSetRegDataEventMask  
someSetRegDataSink  
someSetRegDataSinkMask  
someSetRegDataTimerCount  
someSetRegDataTimerInterval

### Overriding Methods

somInit  
somUnInit

## someClearRegData Method

### Purpose

Clears the registration data.

### IDL Syntax

```
void someClearRegData ( );
```

### Description

This method initializes all fields of a RegData object to their default values.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMRegisterData</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

None.

### Original Class

**SOMEEMRegisterData**

## someSetRegDataClientType Method

### Purpose

Sets the type name for a client event.

### IDL Syntax

```
void someSetRegDataClientType (  
                                in string clientType);
```

### Description

Client events are defined, created, processed, and destroyed entirely by the application. The application can queue several types of client events with EMan. This method sets the client event type field of the registration data object. Thus, this information is communicated to EMan, helping it deal with enqueueing and dequeing the different client events.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMRegisterData</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>clientType</i>	A null-terminated character string identifying the client event type. The contents of this string are entirely up to the user. However, while using class libraries that also use client events, one must make sure that there are no name collisions.

### Return Value

None.

### Original Class

**SOMEEMRegisterData**

### Related Information

Methods: **someClearRegData**



## someSetRegDataEventMask Method

### Purpose

Sets the generic event mask within the registration data using NULL terminated event type list.

### IDL Syntax

```
void someSetRegDataEventMask (
                                in long eventType,
                                in va_list ap);
```

### Description

This allows setting the event mask within the registration data object. Essentially, this tells EMan what kind of event is being registered with it. The event type list is a series of constants defined in eventmsk.h. Although the current interface supports a NULL terminated list of event types, currently each registration with EMan names only one event type. Thus, one usually gives only one named constant as the event type and follows it with a NULL parameter (see example below).

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMRegisterData</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>eventType</i>	A bit constant indicating the type of event being registered with EMan.
<i>ap</i>	Additional event types (usually NULL).

### Return Value

None.

### Example

```
#include <eman.h>
long regId1;
int msgsock;

...
/* Register msgsock socket with EMan for further communication */
_someClearRegData( data, env );
_someSetRegDataEventMask( data, env, EMSinkEvent, NULL );
/* The above call enables EMan to know (during registration) that
we are talking about a Sink Event */
_someSetRegDataSink( data, env, msgsock );
_someSetRegDataSinkMask( data, env, EMInputReadMask);

regId = _someRegisterProc( some_gEMan, env, data,
                          ReadSocketAndPrint, "READMSG" );
```

### Original Class

**SOMEEMRegisterData**

### Related Information

Methods: **someSetRegDataSink**, **someClearRegData**

## someSetRegDataSink Method

### Purpose

Sets the file descriptor (or socket ID, or message queue ID) for the sink event.

### IDL Syntax

```
void someSetRegDataSink (  
                                in long sink);
```

### Description

This method enables setting the true type of an event object. Typically, a subclass of Event calls this method (or overrides this method) to set the event type to indicate its true class(type).

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMRegisterData</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>sink</i>	An integer value indicating the file descriptor for input/output. It can also be a socket ID, pipe ID or a message queue ID.

### Return Value

None.

### Original Class

**SOMEEMRegisterData**

### Related Information

Methods: **someClearRegData**

## someSetRegDataSinkMask Method

### Purpose

Sets the sink mask within the registration data object.

### IDL Syntax

```
void someSetRegDataSinkMask (
                                in unsigned long sinkmask);
```

### Description

The sink mask within the registration data allows one to express interest in different events of the same event source. For example, using this mask one can express interest in being notified when there is input for reading, when the resource is ready for writing output, or just when exceptions occur.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMRegisterData</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>sinkmask</i>	A bit mask indicating the types of events of interest on a given sink.

### Return Value

None.

### Example

```
#include <eman.h>
long regId1;
int msgsock;

...
/* Register msgsock socket with EMan for further communication */
_someClearRegData( data, env );
_someSetRegDataEventMask( data, env, EMSinkEvent, NULL );
_someSetRegDataSink( data, env, msgsock );
_someSetRegDataSinkMask( data, env,
                          EMInputReadMask|EMInputExceptMask);
/* The above call expresses interest in knowing when there is
   input to be read from the socket and when there is an exception
   condition associated with this socket. */
regId = _someRegisterProc( some_gEMan, env, data,
                          ReadSocketAndPrint, "READMSG" );
```

### Original Class

**SOMEEMRegisterData**

### Related Information

Methods: **someSetRegDataSink**, **someClearRegData**

## someSetRegDataTimerCount Method

### Purpose

Sets the number of times the timer will trigger, within the registration data.

### IDL Syntax

```
void someSetRegDataTimerCount (  
                                in long count);
```

### Description

The **someSetRegDataTimerCount** method sets the number of times the timer will trigger, within the registration data. The default behavior is for the timer to trigger indefinitely.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMRegisterData</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>count</i>	An integer indicating the number of times the timer event has to occur.

### Return Value

None.

### Example

```
#include <eman.h>  
long regId1;  
  
...  
/* Register a timer */  
_someClearRegData( data, env );  
_someSetRegDataEventMask( data, env, EMTimerEvent, NULL );  
_someSetRegDataTimerInterval( data, env, 100 );  
_someSetRegDataTimerCount(data, env, 1);  
/* make this a one time timer event */  
regId1 = _someRegister( some_gEMan, env, data, target,  
                       "eventMethod", "Timer 100" );
```

### Original Class

**SOMEEMRegisterData**

### Related Information

Methods: **someClearRegData**

## someSetRegDataTimerInterval Method

### Purpose

Sets the timer interval within the registration data.

### IDL Syntax

```
void someSetRegDataTimerInterval (
    in long interval);
```

### Description

This call allows setting the timer interval (in milliseconds) within the registration data object.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEMRegisterData</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>interval</i>	An integer indicating the timer interval in milliseconds.

### Return Value

None.

### Example

```
#include <eman.h>
long regId1;

...
/* Register a timer */
_someClearRegData( data, env );
_someSetRegDataEventMask( data, env, EMTimerEvent, NULL );
_someSetRegDataTimerInterval( data, env, 100 );
/* Sets the timer interval to 100 milliseconds */
regId1 = _someRegister( some_gEMan, env, data, target,
    "eventMethod", "Timer 100" );
```

### Original Class

**SOMEEMRegisterData**

### Related Information

Methods: **someClearRegData**

---

## SOMEEvent Class

### Description

This is the base class for all generic events within the Event Manager. It simply timestamps an event before it is passed to a callback routine. The event type is set to the true type by a subclass. The types currently used by the Event Management Framework are defined in eventmsk.h. Any subclass of this class must avoid name and value collisions with eventmsk.h.

### File Stem

event

### Base

SOMObject

### Metaclass

SOMClass

### Ancestor Classes

SOMObject

### New Methods

somevGetEventTime  
somevGetEventType  
somevSetEventTime  
somevSetEventType

### Overriding Methods

somlnit

## somevGetEventTime Method

### Purpose

Returns the time of the generic event in milliseconds.

### IDL Syntax

```
unsigned long somevGetEventTime ( );
```

### Description

Eman timestamps every event before dispatching it. The current time is obtained from the operating system (for example, using a 'gettimeofday' call), is converted to milliseconds, and is given as the value of the timestamp. When this function is called, the event timestamp is returned.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

An event timestamp in milliseconds.

### Original Class

**SOMEEvent**

### Related Information

Methods: **somevSetEventTime**

SOMEEvent class

## somevGetEventType Method

### Purpose

Returns the type of the generic event.

### IDL Syntax

```
unsigned long somevGetEventType ( );
```

### Description

This method returns the true type of a given event object (for example, to identify the particular subclass of the event object). The type is an integer valued constant defined in eventmsk.h.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

A type value (an integer constant defined in eventmsk.h).

### Original Class

**SOMEEvent**

### Related Information

Methods: **somevSetEventType**



## somevSetEventTime Method

### Purpose

Sets the time of the generic event (time is in milliseconds).

### IDL Syntax

```
void somevSetEventTime (  
    in unsigned long time);
```

### Description

EMan timestamps every event before dispatching it. The current time is obtained from the operating system (for example, using a 'gettimeofday' call), converted to milliseconds, and is given as the value of the timestamp. When an event occurs, EMan sets the timestamp of the event by calling this method.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>time</i>	The time of day expressed in milliseconds.

### Return Value

None.

### Original Class

**SOMEEvent**

### Related Information

Methods: **somevGetEventTime**

SOMEEvent class

## somevSetEventType Method

### Purpose

Sets the type of the generic event.

### IDL Syntax

```
void somevSetEventType (  
    in unsigned long type);
```

### Description

This method enables setting the true type of an event object. Typically, a subclass of **SOMEEvent** calls this method (or overrides this method) to set the event type to indicate its true type.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMEEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>type</i>	An integer value indicating the type of the event (a constant defined in eventmsk.h).

### Return Value

None.

### Original Class

**SOMEEvent**

### Related Information

Methods: **somevGetEventType**

---

## SOMESinkEvent Class

### Description

This class describes a sink event that is generated by EMan when it notices activity on a registered sink. On AIX, a sink refers to any file descriptor (file open for reading or writing), any pipe descriptor, a socket ID or a message queue ID. On OS/2 or Windows, a sink refers to a socket ID. One can register for three types of interest in a sink: Read interest, Write interest, and Exception interest. (See `eventmsk.h` file to determine the appropriate bit constants and see method **`someSetRegDataSinkMask`** for their use.)

EMan passes an instance of this class as a parameter to the callback registered for Sink Events. The callback can query the instance for some information on the sink.

### File Stem

`sinkev`

### Base

`SOMEEvent`

### Metaclass

`SOMClass`

### Ancestor Classes

`SOMEEvent`, `SOMObject`

### New Methods

`somevGetEventSink`  
`somevSetEventSink`

### Overriding Methods

`somlnit`

SOMESinkEvent class

## somevGetEventSink Method

### Purpose

Returns the sink, or source of I/O, of the generic sink event.

### IDL Syntax

```
long somevGetEventSink ( );
```

### Description

The sink ID in the SinkEvent is returned. For message queues it is the queue ID, for files it is the file descriptor, for sockets it is the socket ID, and for pipes it is the pipe descriptor.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMESinkEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

An integer value indicating the file descriptor for input/output. It can also be a socket ID, pipe ID or a message queue ID.

### Original Class

**SOMESinkEvent**

### Related Information

Methods: **somevSetEventSink**

## somevSetEventSink Method

### Purpose

Sets the sink, or source of I/O, of the generic sink event.

### IDL Syntax

```
void somevSetEventSink (  
    in long sink);
```

### Description

The sink ID in the SinkEvent is set. For message queues, it is the queue ID; for files it is the file descriptor; for sockets it is the socket ID; and for pipes it is the pipe descriptor.

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMESinkEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>sink</i>	An integer value indicating the file descriptor for input/output. It can also be a socket ID, pipe ID, or a message queue ID.

### Return Value

None.

### Original Class

SOMESinkEvent

### Related Information

Methods: somevGetEventSink

---

## SOMTimerEvent Class

### Description

This class describes a timer event that is generated by EMan when any of its registered timers pops.

EMan passes an instance of this class as a parameter to the callbacks registered for Timer Events. The callback can query the instance for information on the timer interval and on any generic event properties.

### File Stem

timerev

### Base

SOMEEvent

### Metaclass

SOMClass

### Ancestor Classes

SOMEEvent, SOMObject

### New Methods

somevGetEventInterval  
somevSetEventInterval

### Overriding Methods

somlInit

## somevGetEventInterval Method

### Purpose

Returns the interval of the generic timer event (time in milliseconds).

### IDL Syntax

```
void somevGetEventInterval ( );
```

### Description

The **somevGetEventInterval** method returns the interval of the generic timer event (time in milliseconds).

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMTimerEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.

### Return Value

The interval time in milliseconds.

### Original Class

**SOMTimerEvent**

### Related Information

Methods: **somevSetEventInterval**

## somevSetEventInterval Method

### Purpose

Sets the interval of the generic timer event (in milliseconds).

### IDL Syntax

```
void somevSetEventInterval (  
                                in long interval);
```

### Description

The **somevSetEventInterval** method sets the interval of the generic timer event (in milliseconds).

### Parameters

<i>receiver</i>	A pointer to an object of class <b>SOMTimerEvent</b> .
<i>ev</i>	A pointer to the <b>Environment</b> structure for the calling method.
<i>interval</i>	The timer interval in milliseconds.

### Return Value

None.

### Original Class

**SOMTimerEvent**

### Related Information

**Methods:** somevGetEventInterval



---

## SOMEWorkProcEvent Class

### Description

This class describes a work procedure event object. It currently has no methods of its own. However, it sets the event type in its super class to say “EMWorkProcEvent” to help identify itself. These events are created and dispatched by EMan when a work procedure (something that the application wants to run when no other events are happening) is registered with EMan.

EMan passes an instance of this class as a parameter to the callback registered for WorkProc Events.

### File Stem

workprev

### Base

SOMEEvent

### Metaclass

SOMClass

### Ancestor Classes

SOMEEvent, SOMObject

### New Methods

None.

### Overriding Methods

somlnit



# Index

## A

activate\_impl\_failed method, Ref–288  
add\_arg method, Ref–233  
add\_class\_to\_impldef method, Ref–197  
add\_impldef method, Ref–198  
add\_item method, Ref–209  
AttributeDef class, Ref–298  
*See also* “Interface Repository Framework”

## B

base\_interfaces attribute, Ref–315  
Before/After methods. *See* “Metaclass Framework, SOMMBeforeAfter metaclass”  
BOA class, Ref–174  
*See also* “DSOM Framework”

## C

change\_id method, Ref–289  
change\_implementation method, Ref–175  
ConstantDef class, Ref–299  
*See also* “Interface Repository Framework”  
Contained class, Ref–300  
*See also* “Interface Repository Framework”  
Container class, Ref–306  
*See also* “Interface Repository Framework”  
contents method, Ref–307  
Context class, Ref–186  
*See also* “DSOM Framework”  
Context\_delete macro, Ref–171  
contexts attribute, Ref–320  
create method, Ref–176  
create\_child method, Ref–187  
create\_constant method, Ref–290  
create\_list method, Ref–226  
create\_operation\_list method, Ref–227  
create\_request method, Ref–253  
create\_request\_args method, Ref–256  
create\_SOM\_ref method, Ref–292

## D

deactivate\_impl method, Ref–178  
deactivate\_obj method, Ref–179  
defined\_in attribute, Ref–300  
delete\_impldef method, Ref–199  
delete\_values method, Ref–188  
describe method, Ref–302  
describe\_contents method, Ref–309

describe\_interface method, Ref–316  
destroy method (Context object), Ref–189  
destroy method (Request object), Ref–235  
dispose method, Ref–180

## DSOM Framework, Ref–159

BOA class, Ref–174  
change\_implementation method, Ref–175  
create method, Ref–176  
deactivate\_impl method, Ref–178  
deactivate\_obj method, Ref–179  
dispose method, Ref–180  
get\_id method, Ref–181  
get\_principal method, Ref–182  
impl\_is\_ready method, Ref–183  
obj\_is\_ready method, Ref–184  
set\_exception method, Ref–185

## Context class, Ref–186

create\_child method, Ref–187  
delete\_values method, Ref–188  
destroy method (Context object), Ref–189  
get\_values method, Ref–190  
set\_one\_value method, Ref–192  
set\_values method, Ref–193

## Functions

get\_next\_response function, Ref–161  
ORBfree function, Ref–162  
send\_multiple\_requests function, Ref–163  
somedExceptionFree function, Ref–165  
SOMD\_Init function, Ref–166  
SOMD\_NoORBfree function, Ref–167  
SOMD\_RegisterCallback function, Ref–168  
SOMD\_Uninit function, Ref–170

## ImplementationDef class, Ref–194

impl\_alias attribute, Ref–194  
impl\_flags attribute, Ref–194  
impl\_hostname attribute, Ref–195  
impl\_id attribute, Ref–194  
impl\_program attribute, Ref–194  
impl\_refdata\_bkup attribute, Ref–195  
impl\_refdata\_file attribute, Ref–195  
impl\_server\_class attribute, Ref–195

## ImplRepository class, Ref–196

add\_class\_to\_impldef method, Ref–197  
add\_impldef method, Ref–198  
delete\_impldef method, Ref–199  
find\_all\_impldefs method, Ref–200  
find\_classes\_by\_impldef method, Ref–201  
find\_impldef method, Ref–202  
find\_impldef\_by\_alias method, Ref–203  
find\_impldef\_by\_class method, Ref–204  
remove\_class\_from\_all method, Ref–205  
remove\_class\_from\_impldef method, Ref–206  
update\_impldef method, Ref–207

## Macros

Context\_delete macro, Ref–171  
Request\_delete macro, Ref–172

## NVList class, Ref–208

add\_item method, Ref–209  
free method, Ref–211  
free\_memory method, Ref–212  
get\_count method, Ref–214  
get\_item method, Ref–215  
set\_item method, Ref–217

## DSOM Framework (cont'd.)

ObjectMgr class, Ref-219  
  somdDestroyObject method, Ref-220  
  somdGetIdFromObject method, Ref-221  
  somdGetObjectFromId method, Ref-222  
  somdNewObject method, Ref-223  
  somdReleaseObject method, Ref-224

ORB class, Ref-225  
  create\_list method, Ref-226  
  create\_operation\_list method, Ref-227  
  get\_default\_context method, Ref-228  
  object\_to\_string method, Ref-229  
  string\_to\_object method, Ref-230

Principal class, Ref-231  
  hostName attribute, Ref-231  
  userName attribute, Ref-231

Request class, Ref-232  
  add\_arg method, Ref-233  
  destroy method (Request object), Ref-235  
  get\_response method, Ref-237  
  invoke method, Ref-239  
  send method, Ref-241

SOMDClientProxy class, Ref-243  
  somdProxyFree method, Ref-244  
  somdProxyGetClass method, Ref-245  
  somdProxyGetClassName method, Ref-246  
  somdReleaseResources method, Ref-247  
  somdTargetFree method, Ref-249  
  somdTargetGetClass method, Ref-250  
  somdTargetGetClassName method, Ref-251

SOMDObject class, Ref-252  
  create\_request method, Ref-253  
  create\_request\_args method, Ref-256  
  duplicate method, Ref-258  
  get\_implementation method, Ref-259  
  get\_interface method, Ref-260  
  is\_constant method, Ref-261  
  is\_nil method, Ref-262  
  is\_proxy method, Ref-263  
  is\_SOM\_ref method, Ref-264  
  release method, Ref-265

SOMDObjectMgr class, Ref-266  
  somd21somFree attribute, Ref-266  
  somdFindAnyServerByClass method, Ref-267  
  somdFindServer method, Ref-268  
  somdFindServerByName method, Ref-269  
  somdFindServersByClass method, Ref-270

SOMDServer class, Ref-271  
  somdCreateObj method, Ref-272  
  somdDeleteObj method, Ref-273  
  somdDispatchMethod method, Ref-274  
  somdGetClassObj method, Ref-275  
  somdObjReferencesCached method, Ref-276  
  somdRefFromSOMObj method, Ref-277  
  somdSOMObjFromRef method, Ref-278

SOMDServerMgr class, Ref-279  
  somdDisableServer method, Ref-280  
  somdEnableServer method, Ref-281  
  somdIsServerEnabled method, Ref-282  
  somdListServer method, Ref-283  
  somdRestartServer method, Ref-284  
  somdShutdownServer method, Ref-285  
  somdStartServer method, Ref-286

## DSOM Framework (cont'd.)

SOMOA class, Ref-287  
  activate\_impl\_failed method, Ref-288  
  change\_id method, Ref-289  
  create\_constant method, Ref-290  
  create\_SOM\_ref method, Ref-292  
  execute\_next\_request method, Ref-293  
  execute\_request\_loop method, Ref-294  
  get\_SOM\_object method, Ref-296

duplicate method, Ref-258

## E

EMan, Ref-513

*See also* "Event Management Framework"

## Event Management Framework, Ref-513

SOMEClientEvent class, Ref-514  
  somevGetEventClientData method, Ref-515  
  somevGetEventClientType method, Ref-516  
  somevSetEventClientData method, Ref-517  
  somevSetEventClientType method, Ref-518

SOMEEMan class, Ref-519  
  someChangeRegData method, Ref-521  
  someGetEManSem method, Ref-522  
  someProcessEvent method, Ref-523  
  someProcessEvents method, Ref-524  
  someQueueEvent method, Ref-525  
  someRegister method, Ref-526  
  someRegisterEv method, Ref-528  
  someRegisterProc method, Ref-530  
  someReleaseEManSem method, Ref-531  
  someShutdown method, Ref-532  
  someUnRegister method, Ref-533

SOMEEMRegisterData class, Ref-534  
  someClearRegData method, Ref-535  
  someSetRegDataClientType method, Ref-536  
  someSetRegDataEventMask method, Ref-537  
  someSetRegDataSink method, Ref-538  
  someSetRegDataSinkMask method, Ref-539  
  someSetRegDataTimerCount method, Ref-540  
  someSetRegDataTimerInterval method, Ref-541

SOMEEvent class, Ref-542  
  somevGetEventTime method, Ref-543  
  somevGetEventType method, Ref-544  
  somevSetEventTime method, Ref-545  
  somevSetEventType method, Ref-546

SOMESinkEvent class, Ref-547  
  somevGetEventSink method, Ref-548  
  somevSetEventSink method, Ref-549

SOMETimerEvent class, Ref-550  
  somevGetEventInterval method, Ref-551  
  somevSetEventInterval method, Ref-552

SOMEWorkProcEvent class, Ref-553

ExceptionDef class, Ref-313  
*See also* "Interface Repository Framework"

execute\_next\_request method, Ref-293  
execute\_request\_loop method, Ref-294

## F

find\_all\_impldefs method, Ref-200  
find\_classes\_by\_impldef method, Ref-201  
find\_impldef method, Ref-202

find\_impldef\_by\_alias method, Ref-203  
find\_impldef\_by\_class method, Ref-204  
free method, Ref-211  
free\_memory method, Ref-212

## G

get\_count method, Ref-214  
get\_default\_context method, Ref-228  
get\_id method, Ref-181  
get\_implementation method, Ref-259  
get\_interface method, Ref-260  
get\_item method, Ref-215  
get\_next\_response function, Ref-161  
get\_principal method, Ref-182  
get\_response method, Ref-237  
get\_SOM\_object method, Ref-296  
get\_values method, Ref-190

## H

hostName attribute, Ref-231

## I

id attribute, Ref-300  
impl\_alias attribute, Ref-194  
impl\_flags attribute, Ref-194  
impl\_hostname attribute, Ref-195  
impl\_id attribute, Ref-194  
impl\_program attribute, Ref-194  
impl\_refdata\_bkup attribute, Ref-195  
impl\_refdata\_file attribute, Ref-195  
impl\_server\_class attribute, Ref-195  
ImplementationDef class, Ref-194  
    *See also* “DSOM Framework”  
impl\_is\_ready method, Ref-183  
ImplRepository class, Ref-196  
    *See also* “DSOM Framework”  
instanceData attribute, Ref-315  
**Interface Repository Framework**, Ref-297  
    AttributeDef class, Ref-298  
        mode attribute, Ref-298  
        type attribute, Ref-298  
    ConstantDef class, Ref-299  
        type attribute, Ref-299  
        value attribute, Ref-299  
    Contained class, Ref-300  
        defined\_in attribute, Ref-300  
        describe method, Ref-302  
        id attribute, Ref-300  
        name attribute, Ref-300  
        somModifiers attribute, Ref-300  
        within method, Ref-304

## Interface Repository Framework (cont'd.)

    Container class, Ref-306  
        contents method, Ref-307  
        describe\_contents method, Ref-309  
        lookup\_name method, Ref-311  
    ExceptionDef class, Ref-313  
        type attribute, Ref-313  
    Functions. *See* “Interface Repository Framework, TypeCode... functions”  
    InterfaceDef class, Ref-314  
        base\_interfaces attribute, Ref-315  
        describe\_interface method, Ref-316  
        instanceData attribute, Ref-315  
    ModuleDef class, Ref-318  
    OperationDef class, Ref-319  
        contexts attribute, Ref-320  
        mode attribute, Ref-319  
        result attribute, Ref-319  
    ParameterDef class, Ref-321  
        mode attribute, Ref-321  
        type attribute, Ref-321  
    Repository class, Ref-322  
        lookup\_id method, Ref-323  
        lookup\_modifier method, Ref-324  
        release\_cache method, Ref-326  
    TypeCode... functions  
        TypeCode\_alignment function, Ref-328  
        TypeCode\_copy function, Ref-329  
        TypeCode\_equal function, Ref-330  
        TypeCode\_free function, Ref-331  
        TypeCode\_kind function, Ref-332  
        TypeCodeNew function, Ref-334  
        TypeCode\_param\_count function, Ref-336  
        TypeCode\_parameter function, Ref-337  
        TypeCode\_print function, Ref-339  
        TypeCode\_setAlignment function, Ref-340  
        TypeCode\_size function, Ref-341  
    TypeDef class, Ref-327  
        type attribute, Ref-327  
    InterfaceDef class, Ref-314  
        *See also* “Interface Repository Framework”  
    invoke method, Ref-239  
    is\_constant method, Ref-261  
    is\_nil method, Ref-262  
    is\_proxy method, Ref-263  
    is\_SOM\_ref method, Ref-264

## L

lookup\_id method, Ref-323  
lookup\_modifier method, Ref-324  
lookup\_name method, Ref-311

## M

M\_SOMPPersistentObject class, Ref-344  
    *See also* “Persistence Framework”  
**Metaclass Framework**, Ref-499  
    SOMMBeforeAfter metaclass, Ref-500  
        sommAfterMethod method, Ref-501  
        sommBeforeMethod method, Ref-503  
    SOMMSingleInstance metaclass, Ref-505  
        sommGetSingleInstance method, Ref-506

## Metaclass Framework (cont'd.)

SOMMTraced metaclass, Ref-507  
sommTracelsOn attribute, Ref-507  
SOMRRReplicable metaclass, Ref-508  
SOMRRReplicableObject class, Ref-509  
somerLoggingType method, Ref-510  
somerReplicableExemptMethod method, Ref-511  
mode attribute, Ref-298, Ref-319, Ref-321  
ModuleDef class, Ref-318  
*See also* "Interface Repository Framework"

## N

name attribute, Ref-300  
NVList class, Ref-208  
*See also* "DSOM Framework"

## O

ObjectMgr class, Ref-219  
*See also* "DSOM Framework"  
object\_to\_string method, Ref-229  
obj\_is\_ready method, Ref-184  
OperationDef class, Ref-319  
*See also* "Interface Repository Framework"  
ORB class, Ref-225  
*See also* "DSOM Framework"  
ORBfree function, Ref-162

## P

ParameterDef class, Ref-321  
*See also* "Interface Repository Framework"

## Persistence Framework, Ref-343

M\_SOMPPersistentObject class, Ref-344  
sompGetClassLevelEncoderDecoderName method, Ref-345  
sompSetClassLevelEncoderDecoderName method, Ref-346  
SOMPAscii class, Ref-347  
SOMPAsciiMediaInterface class, Ref-348  
sompGetMediaName method, Ref-349  
sompInitSpecific method, Ref-350  
sompQueryBlockSize method, Ref-352  
sompStat method, Ref-353  
SOMPAttrEncoderDecoder class, Ref-354  
SOMPBinary class, Ref-355  
SOMPBinaryFileMedia class, Ref-356  
SOMPDecoderDecoderAbstract class, Ref-357  
sompEDRead method, Ref-358  
sompEDWrite method, Ref-359  
SOMPFileMediaAbstract class, Ref-360  
sompGetOffset method, Ref-361  
sompInitReadOnly method, Ref-362  
sompInitReadWrite method, Ref-363  
sompReadBytes method, Ref-364  
sompReadCharacter method, Ref-365  
sompReadDouble method, Ref-366  
sompReadFloat method, Ref-367  
sompRead<IntegralType> methods, Ref-368  
sompReadLine method, Ref-369

## Persistence Framework (cont'd.)

SOMPFileMediaAbstract class (cont'd.)  
sompReadOctet method, Ref-370  
sompReadSomobject method, Ref-371  
sompReadString method, Ref-372  
sompReadStringToBuffer method, Ref-373  
sompReadTypeCode method, Ref-374  
sompSeekPosition method, Ref-375  
sompSeekPositionRel method, Ref-375  
sompWriteBytes method, Ref-376  
sompWriteCharacter method, Ref-377  
sompWriteDouble method, Ref-378  
sompWriteFloat method, Ref-379  
sompWrite<IntegralType> methods, Ref-380  
sompWriteLine method, Ref-381  
sompWriteOctet method, Ref-382  
sompWriteSomobject method, Ref-383  
sompWriteString method, Ref-384  
sompWriteTypeCode method, Ref-385  
SOMPIdAssigner class, Ref-386  
SOMPIdAssignerAbstract class, Ref-387  
sompGetSystemAssignedId method, Ref-388  
SOMPIOGroup class, Ref-389  
sompAddToGroup method, Ref-390  
sompCount method, Ref-392  
sompFindByKey method, Ref-394  
sompFirst method, Ref-396  
sompFreeliterator method, Ref-398  
sompNewIterator method, Ref-399  
sompNextObjectInGroup method, Ref-400  
sompRemoveFromGroup method, Ref-401  
SOMPIOGroupMgrAbstract class, Ref-403  
sompDeleteObjectFromGroup method, Ref-404  
sompFreeMediaInterface method, Ref-405  
sompGetMediaInterface method, Ref-406  
sompGroupExists method, Ref-407  
sompInstantiateMediaInterface method, Ref-408  
sompMediaFormatOk method, Ref-409  
sompNewMediaInterface method, Ref-410  
sompObjectInGroup method, Ref-411  
sompReadGroup method, Ref-412  
sompReadObjectData method, Ref-413  
sompWriteGroup method, Ref-414  
SOMPMediaInterfaceAbstract class, Ref-415  
sompClose method, Ref-416  
sompOpen method, Ref-417  
SOMPPersistentId class, Ref-418  
sompEqualsIOGroupName method, Ref-419  
sompGetGroupOffset method, Ref-420  
sompGetIOGroupMgrClassName method, Ref-421  
sompGetIOGroupMgrClassNameLen method, Ref-422  
sompGetIOGroupName method, Ref-423  
sompGetIOGroupNameLen method, Ref-424  
sompSetGroupOffset method, Ref-425  
sompSetIOGroupMgrClassName method, Ref-426  
sompSetIOGroupName method, Ref-427  
SOMPPersistentObject class, Ref-428  
sompActivated method, Ref-429  
sompCheckState method, Ref-430  
sompClearState method, Ref-431  
sompEquals method, Ref-432  
sompFreeEncoderDecoder method, Ref-433  
sompGetDirty method, Ref-434  
sompGetEncoderDecoder method, Ref-435

## Persistence Framework (cont'd.)

- SOMPPersistentObject class (cont'd.)
  - sompGetEncoderDecoderName method, Ref-436
  - sompGetIOGroup method, Ref-437
  - sompGetPersistentId method, Ref-438
  - sompGetPersistentIdString method, Ref-439
  - sompGetRelativeIdString method, Ref-440
  - somplnitGivenId method, Ref-441
  - somplnitIOGroup method, Ref-442
  - somplnitNearObject method, Ref-443
  - somplnitNextAvail method, Ref-444
  - somplsDirty method, Ref-445
  - sompMarkForCompaction method, Ref-446
  - sompPassivate method, Ref-447
  - sompSetDirty method, Ref-448
  - sompSetEncoderDecoderName method, Ref-449
  - sompSetState method, Ref-450
- SOMPPersistentStorageMgr class, Ref-451
  - sompAddIdToReadSet method, Ref-452
  - sompAddObjectToWriteSet method, Ref-453
  - sompDeleteObject method, Ref-454
  - sompObjectExists method, Ref-455
  - sompRestoreObject method, Ref-456
  - sompRestoreObjectFromIdString method, Ref-457
  - sompRestoreObjectWithoutChildren method, Ref-458
  - sompStoreObject method, Ref-459
  - sompStoreObjectWithoutChildren method, Ref-460
- SOMUTId class, Ref-461
  - somutCompareId method, Ref-462
  - somutEqualsId method, Ref-463
  - somutHashId method, Ref-464
  - somutSetIdId method, Ref-465
- SOMUTStringId class, Ref-466
  - somutCompareString method, Ref-467
  - somutEqualsString method, Ref-468
  - somutGetIdString method, Ref-469
  - somutGetIdStringLength method, Ref-470
  - somutSetIdString method, Ref-471
- Principal class, Ref-231
  - See also* "DSOM Framework"

## R

- release method, Ref-265
- release\_cache method, Ref-326
- remove\_class\_from\_all method, Ref-205
- remove\_class\_from\_impldef method, Ref-206

## Replication Framework, Ref-473

- SOMR class, Ref-474
- SOMRLinearizable class, Ref-475
  - somrGetState method, Ref-476
  - somrSetState method, Ref-477
- SOMRNameable class, Ref-478
  - somrGetObjName method, Ref-479
  - somrSetObjName method, Ref-480
- SOMRReplicable metaclass, Ref-508
  - See also* "Metaclass Framework, SOMR... classes and methods"
- SOMRReplicableObject class, Ref-509

## Replication Framework (cont'd.)

- SOMRReplicbl class, Ref-481
  - somrApplyUpdates method, Ref-483
  - somrDoDirective method, Ref-484
  - somrGetSecurityPolicy method, Ref-485
  - somrLock method, Ref-486
  - somrLockNlogOp method, Ref-487
  - somrPin method, Ref-488
  - somrReleaseLockNAbortOp method, Ref-489
  - somrReleaseLockNAbortUpdate method, Ref-490
  - somrReleaseNPropagateOperation method, Ref-491
  - somrReleaseNPropagateUpdate method, Ref-492
  - somrReplnit method, Ref-494
  - somrRepUninit method, Ref-496
  - somrUnPin method, Ref-497
- Repository class, Ref-322
  - See also* "Interface Repository Framework"
- Request class, Ref-232
  - See also* "DSOM Framework"
- Request\_delete macro, Ref-172
- result attribute, Ref-319

## S

- send method, Ref-241
- send\_multiple\_requests function, Ref-163
- set\_exception method, Ref-185
- set\_item method, Ref-217
- set\_one\_value method, Ref-192
- set\_values method, Ref-193
- SOM kernel**, Ref-1
  - Functions
    - somApply function, Ref-2
    - somBeginPersistentIds function, Ref-4
    - somBuildClass function, Ref-6
    - SOMCalloc function, Ref-40
    - somCheckId function, Ref-7
    - SOMClassInitFuncName function, Ref-41
    - somClassResolve function, Ref-8
    - somCompareIds function, Ref-10
    - somDataResolve function, Ref-11
    - SOMDeleteModule function, Ref-42
    - somEndPersistentIds function, Ref-12
    - somEnvironmentEnd function, Ref-13
    - somEnvironmentNew function, Ref-14
    - SOMError function, Ref-43
    - somExceptionFree function, Ref-15
    - somExceptionId function, Ref-16
    - somExceptionValue function, Ref-17
    - SOMFree function, Ref-44
    - somGetGlobalEnvironment function, Ref-18
    - somIdFromString function, Ref-19
    - SOMInitModule function, Ref-45
    - somIsObj function, Ref-20
    - SOMLoadModule function, Ref-47
    - somLPrintf function, Ref-21
    - somMainProgram function, Ref-22
    - SOMMalloc function, Ref-48
    - SOMOutCharRoutine function, Ref-49
    - somParentNumResolve function, Ref-23
    - somParentResolve function, Ref-25
    - somPrefixLevel function, Ref-26

## SOM kernel (cont'd.)

### Functions (cont'd.)

- somPrintf function, Ref-27
- SOMRealloc function, Ref-50
- somRegisterId function, Ref-28
- somResolve function, Ref-29
- somResolveByName function, Ref-31
- somSetException function, Ref-32
- somSetExpectedIds function, Ref-34
- somSetOutChar function, Ref-35
- somStringFromId function, Ref-36
- somTotalRegIds function, Ref-37
- somUniqueKey function, Ref-38
- somVprintf function, Ref-39

### Macros

- SOM\_Assert macro, Ref-51
- SOM\_ClassLibrary macro, Ref-52
- SOM\_CreateLocalEnvironment macro, Ref-53
- SOM\_DestroyLocalEnvironment macro, Ref-54
- SOM\_Error macro, Ref-55
- SOM\_Expect macro, Ref-56
- SOM\_GetClass macro, Ref-57
- SOM\_InitEnvironment macro, Ref-58
- SOM\_MainProgram macro, Ref-59
- SOM\_NoTrace macro, Ref-60
- SOM\_ParentNumResolve macro, Ref-61
- SOM\_Resolve macro, Ref-62
- SOM\_ResolveNoCheck macro, Ref-63
- SOM\_SubstituteClass macro, Ref-64
- SOM\_Test macro, Ref-65
- SOM\_TestC macro, Ref-66
- SOM\_UninitEnvironment macro, Ref-67
- SOM\_WarnMsg macro, Ref-68

### SOMClass class, Ref-69

- somAddDynamicMethod method, Ref-72
- somAllocate method, Ref-74
- somCheckVersion method, Ref-75
- somClassReady method, Ref-77
- somDeallocate method, Ref-78
- somDescendedFrom method, Ref-79
- somFindMethod(OK) methods, Ref-80
- somFindSMMethod(OK) methods, Ref-82
- somGetInstancePartSize method, Ref-83
- somGetInstanceSize method, Ref-84
- somGetInstanceToken method, Ref-85
- somGetMemberToken method, Ref-86
- somGetMethodData method, Ref-87
- somGetMethodDescriptor method, Ref-88
- somGetMethodIndex method, Ref-89
- somGetMethodToken method, Ref-90
- somGetName method, Ref-91
- somGetNthMethodData method, Ref-92
- somGetNthMethodInfo method, Ref-93
- somGetNumMethods method, Ref-94
- somGetNumStaticMethods method, Ref-95
- somGetParents method, Ref-96
- somGetVersionNumbers method, Ref-97
- somInstanceDataOffsets attribute, Ref-69
- somLookupMethod method, Ref-98
- somNew(NoInit) methods, Ref-100
- somRenew(NoInit) methods, Ref-101
- somSupportsMethod method, Ref-103

## SOM kernel (cont'd.)

### SOMClassMgr class, Ref-104

- somClassFromId method, Ref-106
- somFindClass method, Ref-107
- somFindClsInFile method, Ref-109
- somGetInitFunction method, Ref-111
- somGetRelatedClasses method, Ref-113
- somInterfaceRepository attribute, Ref-104
- somLoadClassFile method, Ref-115
- somLocateClassFile method, Ref-116
- somMergeInto method, Ref-117
- somRegisterClass method, Ref-119
- somRegisteredClasses attribute, Ref-104
- somSubstituteClass method, Ref-120
- somUnloadClassFile method, Ref-122
- somUnregisterClass method, Ref-123

### SOMObject class, Ref-124

- somCastObj method, Ref-126
- somClassDispatch method, Ref-136
- somDefaultAssign method, Ref-127
- somDefaultConstAssign method, Ref-128
- somDefaultConstCopyInit method, Ref-129
- somDefaultCopyInit method, Ref-130
- somDefaultInit method, Ref-132
- somDestruct method, Ref-134
- somDispatch method, Ref-136
- somDispatchX method, Ref-139
- somDumpSelf method, Ref-141
- somDumpSelfInt method, Ref-142
- somFree method, Ref-144
- somGetClass method, Ref-145
- somGetClassName method, Ref-146
- somGetSize method, Ref-147
- somInit method, Ref-148
- somIsA method, Ref-150
- somIsInstanceOf method, Ref-152
- somPrintSelf method, Ref-154
- somResetObj method, Ref-155
- somRespondsTo method, Ref-156
- somUninit method, Ref-157

SOM metaclass classes/methods. *See* "Metaclass Framework"

somAddDynamicMethod method, Ref-72

somAllocate method, Ref-74

somApply function, Ref-2

SOM\_Assert macro, Ref-51

somBeginPersistentIds function, Ref-4

somBuildClass function, Ref-6

SOMCalloc function, Ref-40

somCastObj method, Ref-126

somCheckId function, Ref-7

somCheckVersion method, Ref-75

SOMClass class, Ref-69

*See also* "SOM kernel"

somClassDispatch method, Ref-136

somClassFromId method, Ref-106

SOMClassInitFuncName function, Ref-41

SOM\_ClassLibrary macro, Ref-52

SOMClassMgr class, Ref-104

*See also* "SOM kernel"



somClassReady method, Ref-77  
 somClassResolve function, Ref-8  
 somCompareIds function, Ref-10  
 SOM\_CreateLocalEnvironment macro, Ref-53  
 somd21somFree attribute, Ref-266  
 somDataResolve function, Ref-11  
 SOMDClientProxy class, Ref-243  
     *See also* "DSOM Framework"  
 somdCreateObj method, Ref-272  
 somdDeleteObj method, Ref-273  
 somdDestroyObject method, Ref-220  
 somdDisableServer method, Ref-280  
 somdDispatchMethod method, Ref-274  
 somDeallocate method, Ref-78  
 somDefaultAssign method, Ref-127  
 somDefaultConstAssign method, Ref-128  
 somDefaultConstCopyInit method, Ref-129  
 somDefaultCopyInit method, Ref-130  
 somDefaultInit method, Ref-132  
 SOMDeleteModule function, Ref-42  
 somdEnableServer method, Ref-281  
 somDescendedFrom method, Ref-79  
 SOM\_DestroyLocalEnvironment macro, Ref-54  
 somDestruct method, Ref-134  
 somdExceptionFree function, Ref-165  
 somdFindAnyServerByClass method, Ref-267  
 somdFindServer method, Ref-268  
 somdFindServerByName method, Ref-269  
 somdFindServersByClass method, Ref-270  
 somdGetClassObj method, Ref-275  
 somdGetIdFromObject method, Ref-221  
 somdGetObjectFromId method, Ref-222  
 SOMD\_Init function, Ref-166  
 somDispatch method, Ref-136  
 somDispatchX method, Ref-139  
 somdIsServerEnabled method, Ref-282  
 somdListServer method, Ref-283  
 somdNewObject method, Ref-223  
 SOMD\_NoORBfree function, Ref-167  
 SOMDObject class, Ref-252  
     *See also* "DSOM Framework"  
 SOMDObjectMgr class, Ref-266  
     *See also* "DSOM Framework"  
 somdObjReferencesCached method, Ref-276  
 somdProxyFree method, Ref-244  
 somdProxyGetClass method, Ref-245  
 somdProxyGetClassName method, Ref-246  
 somdRefFromSOMObj method, Ref-277  
 SOMD\_RegisterCallback function, Ref-168  
 somdReleaseObject method, Ref-224  
 somdReleaseResources method, Ref-247  
 somdRestartServer method, Ref-284  
 SOMDServer class, Ref-271  
     *See also* "DSOM Framework"  
 SOMDServerMgr class, Ref-279  
     *See also* "DSOM Framework"  
 somdShutdownServer method, Ref-285  
 somdSOMObjFromRef method, Ref-278  
 somdStartServer method, Ref-286  
 somdTargetFree method, Ref-249  
 somdTargetGetClass method, Ref-250  
 somdTargetGetClassName method, Ref-251  
 somDumpSelf method, Ref-141  
 somDumpSelfInt method, Ref-142  
 SOMD\_Uninit function, Ref-170  
 someChangeRegData method, Ref-521  
 someClearRegData method, Ref-535  
 SOMEClientEvent class, Ref-514  
     *See also* "Event Management Framework"  
 SOMEEMan class, Ref-519  
     *See also* "Event Management Framework"  
 SOMEEMRegisterData class, Ref-534  
     *See also* "Event Management Framework"  
 SOMEEvent class, Ref-542  
     *See also* "Event Management Framework"  
 someGetEManSem method, Ref-522  
 somEndPersistentIds function, Ref-12  
 somEnvironmentEnd function, Ref-13  
 somEnvironmentNew function, Ref-14  
 someProcessEvent method, Ref-523  
 someProcessEvents method, Ref-524  
 someQueueEvent method, Ref-525  
 someRegister method, Ref-526  
 someRegisterEv method, Ref-528  
 someRegisterProc method, Ref-530  
 someReleaseEManSem method, Ref-531  
 SOMError function, Ref-43  
 SOM\_Error macro, Ref-55  
 someSetRegDataClientType method, Ref-536  
 someSetRegDataEventMask method, Ref-537  
 someSetRegDataSink method, Ref-538  
 someSetRegDataSinkMask method, Ref-539  
 someSetRegDataTimerCount method, Ref-540  
 someSetRegDataTimerInterval method, Ref-541  
 someShutdown method, Ref-532  
 SOMESinkEvent class, Ref-547  
     *See also* "Event Management Framework"  
 SOMETimerEvent class, Ref-550  
     *See also* "Event Management Framework"  
 someUnRegister method, Ref-533  
 somevGetEventClientData method, Ref-515  
 somevGetEventClientType method, Ref-516  
 somevGetEventInterval method, Ref-551  
 somevGetEventSink method, Ref-548

somevGetEventTime method, Ref-543  
 somevGetEventType method, Ref-544  
 somevSetEventClientData method, Ref-517  
 somevSetEventClientType method, Ref-518  
 somevSetEventInterval method, Ref-552  
 somevSetEventSink method, Ref-549  
 somevSetEventTime method, Ref-545  
 somevSetEventType method, Ref-546  
 SOMEWorkProcEvent class, Ref-553  
     *See also* "Event Management Framework"  
 somExceptionFree function, Ref-15  
 somExceptionId function, Ref-16  
 somExceptionValue function, Ref-17  
 SOM\_Expect macro, Ref-56  
 somFindClass method, Ref-107  
 somFindClsInFile method, Ref-109  
 somFindMethod(OK) methods, Ref-80  
 somFindSMethod(OK) methods, Ref-82  
 SOMFree function, Ref-44  
 somFree method, Ref-144  
 SOM\_GetClass macro, Ref-57  
 somGetClass method, Ref-145  
 somGetClassName method, Ref-146  
 somGetGlobalEnvironment function, Ref-18  
 somGetInitFunction method, Ref-111  
 somGetInstancePartSize method, Ref-83  
 somGetInstanceSize method, Ref-84  
 somGetInstanceToken method, Ref-85  
 somGetMemberToken method, Ref-86  
 somGetMethodData method, Ref-87  
 somGetMethodDescriptor method, Ref-88  
 somGetMethodIndex method, Ref-89  
 somGetMethodToken method, Ref-90  
 somGetName method, Ref-91  
 somGetNthMethodData method, Ref-92  
 somGetNthMethodInfo method, Ref-93  
 somGetNumMethods method, Ref-94  
 somGetNumStaticMethods method, Ref-95  
 somGetParents method, Ref-96  
 somGetRelatedClasses method, Ref-113  
 somGetSize method, Ref-147  
 somGetVersionNumbers method, Ref-97  
 somIdFromString function, Ref-19  
 somInit method, Ref-148  
 SOM\_InitEnvironment macro, Ref-58  
 SOMInitModule function, Ref-45  
 somInstanceDataOffsets attribute, Ref-69  
 somInterfaceRepository attribute, Ref-104  
 somIsA method, Ref-150  
 somIsInstanceOf method, Ref-152  
 somIsObj function, Ref-20  
 somLoadClassFile method, Ref-115  
 SOMLoadModule function, Ref-47  
 somLocateClassFile method, Ref-116  
 somLookupMethod method, Ref-98  
 somLPrintf function, Ref-21  
 sommAfterMethod method, Ref-501  
 somMainProgram function, Ref-22  
 SOM\_MainProgram macro, Ref-59  
 SOMMalloc function, Ref-48  
 SOMMBeforeAfter metaclass, Ref-500  
     *See also* "Metaclass Framework"  
 sommBeforeMethod method, Ref-503  
 somMergeInto method, Ref-117  
 sommGetSingleInstance method, Ref-506  
 somModifiers attribute, Ref-300  
 SOMMSingleInstance metaclass, Ref-505  
     *See also* "Metaclass Framework"  
 SOMMTraced metaclass, Ref-507  
     *See also* "Metaclass Framework"  
 sommTracelsOn attribute, Ref-507  
 somNew(Nolnit) methods, Ref-100  
 SOM\_NoTrace macro, Ref-60  
 SOMOA class, Ref-287  
     *See also* "DSOM Framework"  
 SOMObject class, Ref-124  
     *See also* "SOM kernel"  
 SOMOutCharRoutine function, Ref-49  
 sompActivated method, Ref-429  
 sompAddIdToReadSet method, Ref-452  
 sompAddObjectToWriteSet method, Ref-453  
 sompAddToGroup method, Ref-390  
 somParentNumResolve function, Ref-23  
 SOM\_ParentNumResolve macro, Ref-61  
 somParentResolve function, Ref-25  
 SOMPAscii class, Ref-347  
     *See also* "Persistence Framework"  
 SOMPAsciiMediaInterface class, Ref-348  
     *See also* "Persistence Framework"  
 SOMPAttrEncoderDecoder class, Ref-354  
     *See also* "Persistence Framework"  
 SOMPBinary class, Ref-355  
     *See also* "Persistence Framework"  
 SOMPBinaryFileMedia class, Ref-356  
     *See also* "Persistence Framework"  
 sompCheckState method, Ref-430  
 sompClearState method, Ref-431  
 sompClose method, Ref-416  
 sompCount method, Ref-392  
 sompDeleteObject method, Ref-454  
 sompDeleteObjectFromGroup method, Ref-404  
 sompEDRead method, Ref-358  
 sompEDWrite method, Ref-359  
 SOMPEncoderDecoderAbstract class, Ref-357  
     *See also* "Persistence Framework"

sompEquals method, Ref-432  
 sompEqualsIOGroupName method, Ref-419  
 SOMPFileMediaAbstract class, Ref-360  
     *See also* "Persistence Framework"  
 sompFindByKey method, Ref-394  
 sompFirst method, Ref-396  
 sompFreeEncoderDecoder method, Ref-433  
 sompFreeIterator method, Ref-398  
 sompFreeMediaInterface method, Ref-405  
 sompGetClassLevelEncoderDecoderName method,  
     Ref-345  
 sompGetDirty method, Ref-434  
 sompGetEncoderDecoder method, Ref-435  
 sompGetEncoderDecoderName method, Ref-436  
 sompGetGroupOffset method, Ref-420  
 sompGetIOGroup method, Ref-437  
 sompGetIOGroupMgrClassName method, Ref-421  
 sompGetIOGroupMgrClassNameLen method, Ref-422  
 sompGetIOGroupName method, Ref-423  
 sompGetIOGroupNameLen method, Ref-424  
 sompGetMediaInterface method, Ref-406  
 sompGetMediaName method, Ref-349  
 sompGetOffset method, Ref-361  
 sompGetPersistentId method, Ref-438  
 sompGetPersistentIdString method, Ref-439  
 sompGetRelativeIdString method, Ref-440  
 sompGetSystemAssignedId method, Ref-388  
 sompGroupExists method, Ref-407  
 SOMPidAssigner class, Ref-386  
     *See also* "Persistence Framework"  
 SOMPidAssignerAbstract class, Ref-387  
     *See also* "Persistence Framework"  
 sompInitGivenId method, Ref-441  
 sompInitIOGroup method, Ref-442  
 sompInitNearObject method, Ref-443  
 sompInitNextAvail method, Ref-444  
 sompInitReadOnly method, Ref-362  
 sompInitReadWrite method, Ref-363  
 sompInitSpecific method, Ref-350  
 sompInstantiateMediaInterface method, Ref-408  
 SOMPIOGroup class, Ref-389  
     *See also* "Persistence Framework"  
 SOMPIOGroupMgrAbstract class, Ref-403  
     *See also* "Persistence Framework"  
 somplsDirty method, Ref-445  
 sompMarkForCompaction method, Ref-446  
 sompMediaFormatOk method, Ref-409  
 SOMPMediaInterfaceAbstract class, Ref-415  
     *See also* "Persistence Framework"  
 sompNewIterator method, Ref-399  
 sompNewMediaInterface method, Ref-410  
 sompNextObjectInGroup method, Ref-400  
 sompObjectExists method, Ref-455  
 sompObjectInGroup method, Ref-411  
 sompOpen method, Ref-417  
 sompPassivate method, Ref-447  
 SOMPPersistentId class, Ref-418  
     *See also* "Persistence Framework"  
 SOMPPersistentObject class, Ref-428  
     *See also* "Persistence Framework"  
 SOMPPersistentStorageMgr class, Ref-451  
     *See also* "Persistence Framework"  
 sompQueryBlockSize method, Ref-352  
 sompReadBytes method, Ref-364  
 sompReadCharacter method, Ref-365  
 sompReadDouble method, Ref-366  
 sompReadFloat method, Ref-367  
 sompReadGroup method, Ref-412  
 sompRead<IntegralType> methods, Ref-368  
 sompReadLine method, Ref-369  
 sompReadObjectData method, Ref-413  
 sompReadOctet method, Ref-370  
 sompReadSomobject method, Ref-371  
 sompReadString method, Ref-372  
 sompReadStringToBuffer method, Ref-373  
 sompReadTypeCode method, Ref-374  
 somPrefixLevel function, Ref-26  
 sompRemoveFromGroup method, Ref-401  
 sompRestoreObject method, Ref-456  
 sompRestoreObjectFromIdString method, Ref-457  
 sompRestoreObjectWithoutChildren method, Ref-458  
 somPrintf function, Ref-27  
 somPrintSelf method, Ref-154  
 sompSeekPosition method, Ref-375  
 sompSeekPositionRel method, Ref-375  
 sompSetClassLevelEncoderDecoderName method,  
     Ref-346  
 sompSetDirty method, Ref-448  
 sompSetEncoderDecoderName method, Ref-449  
 sompSetGroupOffset method, Ref-425  
 sompSetIOGroupMgrClassName method, Ref-426  
 sompSetIOGroupName method, Ref-427  
 sompSetState method, Ref-450  
 sompStat method, Ref-353  
 sompStoreObject method, Ref-459  
 sompStoreObjectWithoutChildren method, Ref-460  
 sompWriteBytes method, Ref-376  
 sompWriteCharacter method, Ref-377  
 sompWriteDouble method, Ref-378  
 sompWriteFloat method, Ref-379  
 sompWriteGroup method, Ref-414  
 sompWrite<IntegralType> methods, Ref-380  
 sompWriteLine method, Ref-381  
 sompWriteOctet method, Ref-382

sompWriteSomobject method, Ref–383  
 sompWriteString method, Ref–384  
 sompWriteTypeCode method, Ref–385  
 SOMR class, Ref–474  
     *See also* “Replication Framework”  
 somrApplyUpdates method, Ref–483  
 somrDoDirective method, Ref–484  
 SOMRealloc function, Ref–50  
 somRegisterClass method, Ref–119  
 somRegisteredClasses attribute, Ref–104  
 somRegisterId function, Ref–28  
 somRenew(Nolnit) methods, Ref–101  
 somResetObj method, Ref–155  
 somResolve function, Ref–29  
 SOM\_Resolve macro, Ref–62  
 somResolveByName function, Ref–31  
 SOM\_ResolveNoCheck macro, Ref–63  
 somRespondsTo method, Ref–156  
 somrGetObjName method, Ref–479  
 somrGetSecurityPolicy method, Ref–485  
 somrGetState method, Ref–476  
 SOMRLinearizable class, Ref–475  
     *See also* “Replication Framework”  
 somrLock method, Ref–486  
 somrLockNlogOp method, Ref–487  
 somrLoggingType method, Ref–510  
 SOMRNameable class, Ref–478  
     *See also* “Replication Framework”  
 somrPin method, Ref–488  
 somrReleaseLockNAbortOp method, Ref–489  
 somrReleaseLockNAbortUpdate method, Ref–490  
 somrReleaseNPropagateOperation method, Ref–491  
 somrReleaseNPropagateUpdate method, Ref–492  
 somrReplnit method, Ref–494  
 SOMRReplicable metaclass, Ref–508  
     *See also* “Metaclass Framework”  
 somrReplicableExemptMethod method, Ref–511  
 SOMRReplicableObject class, Ref–509  
     *See also* “Metaclass Framework”  
 SOMRReplicbl class, Ref–481  
     *See also* “Replication Framework”  
 somrRepUninit method, Ref–496  
 somrSetObjName method, Ref–480  
 somrSetState method, Ref–477  
 somrUnPin method, Ref–497  
 somSetException function, Ref–32  
 somSetExpectedIds function, Ref–34  
 somSetOutChar function, Ref–35  
 somStringFromId function, Ref–36  
 SOM\_SubstituteClass macro, Ref–64

somSubstituteClass method, Ref–120  
 somSupportsMethod method, Ref–103  
 SOM\_Test macro, Ref–65  
 SOM\_TestC macro, Ref–66  
 somTotalRegIds function, Ref–37  
 somUninit method, Ref–157  
 SOM\_UninitEnvironment macro, Ref–67  
 somUniqueKey function, Ref–38  
 somUnloadClassFile method, Ref–122  
 somUnregisterClass method, Ref–123  
 somutCompareId method, Ref–462  
 somutCompareString method, Ref–467  
 somutEqualsId method, Ref–463  
 somutEqualsString method, Ref–468  
 somutGetIdString method, Ref–469  
 somutGetIdStringLength method, Ref–470  
 somutHashId method, Ref–464  
 SOMUTId class, Ref–461  
     *See also* “Persistence Framework”  
 somutSetIdId method, Ref–465  
 somutSetIdString method, Ref–471  
 SOMUTStringId class, Ref–466  
     *See also* “Persistence Framework”  
 somVprintf function, Ref–39  
 SOM\_WarnMsg macro, Ref–68  
 string\_to\_object method, Ref–230

## T

Tracing methods. *See* “Metaclass Framework, SOMMTraced metaclass”  
 type attribute, Ref–298, Ref–299, Ref–313, Ref–321, Ref–327  
 TypeCode\_alignment function, Ref–328  
 TypeCode\_copy function, Ref–329  
 TypeCode\_equal function, Ref–330  
 TypeCode\_free function, Ref–331  
 TypeCode\_kind function, Ref–332  
 TypeCodeNew function, Ref–334  
 TypeCode\_param\_count function, Ref–336  
 TypeCode\_parameter function, Ref–337  
 TypeCode\_print function, Ref–339  
 TypeCode\_setAlignment function, Ref–340  
 TypeCode\_size function, Ref–341  
 TypeDef class, Ref–327  
     *See also* “Interface Repository Framework”

## U

update\_impldef method, Ref–207  
 userName attribute, Ref–231  
 Utility metaclasses. *See* “Metaclass Framework”

## V

value attribute, Ref–299

## W

within method, Ref–304