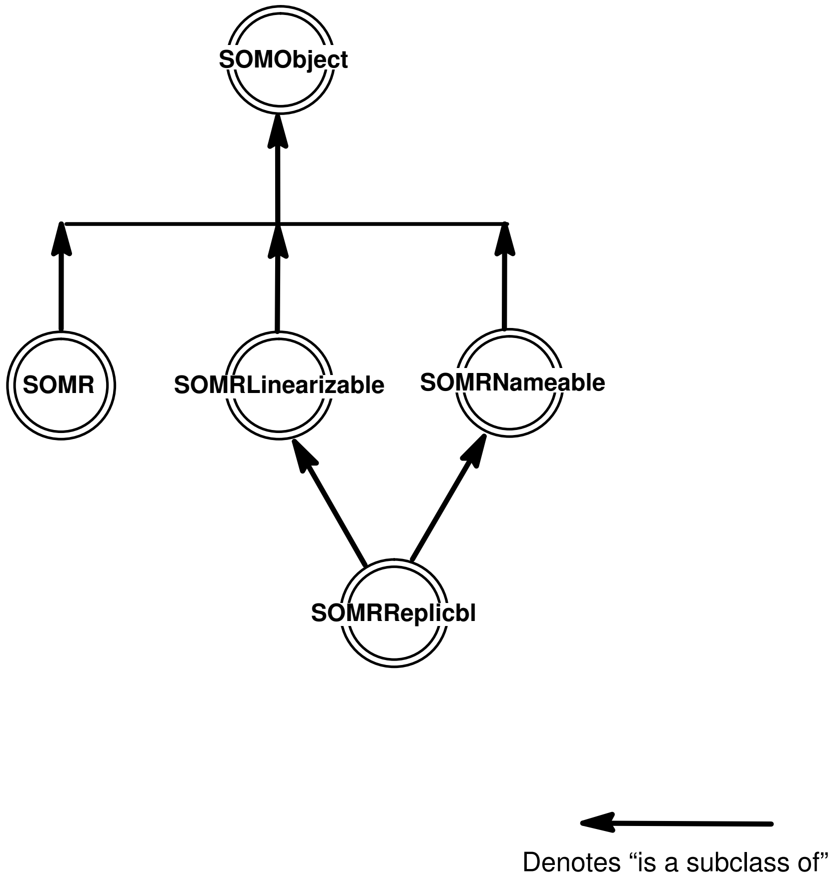
Replication Framework Reference

Replication Framework Class Organization



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

somInit

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

somInit

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

receiver A pointer to an object of class **SOMRLinearizable**.

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

buf 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

receiver	A pointer to an object of class **SOMRLinearizable**.
ev	A pointer to the **Environment** structure for the caller.
buf	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

somInit
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

receiver	A pointer to an object of class **SOMRNameable**.
ev	A pointer to the **Environment** 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

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

receiver	A pointer to an object of class **SOMRNameable**.
ev	A pointer to the **Environment** structure for the caller.
name	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

somrPin
somrReleaseLockNAbortOp
somrReleaseLockNAbortUpdate
somrReleaseNPropagateUpdate
somrReleaseNPropagateOperation
somrRepInit
somrRepUninit
somrUnpin

Overriding Methods

somInit
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

receiver A pointer to an object of class **SOMRReplicbl**.

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

buffer A pointer to a character buffer representing update information.

bufferLen The size of *buffer*.

objIntId 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

receiver A pointer to an object of class **SOMRReplicbl**.

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

str 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: somrRepInit, somrRepUninit

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

receiver A pointer to an object of class **SOMRReplicbl**.

ev A pointer to the **Environment** 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 **somrRepInit**). The exception raised indicates whether the lock was successfully obtained.

Parameters

receiver	A pointer to an object of class **SOMRReplicbl**.
ev	A pointer to the **Environment** 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 **somrRepInit**). This method is the same as **somrLock**, but has the additional responsibility of logging the method that is requesting the lock.

Parameters

receiver A pointer to an object of class **SOMRReplicbl**.

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

classname The name of the class of the object.

methodname The name of the method to be logged.

ap A pointer to a **va_list** that specifies the arguments with which methodName 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

receiver	A pointer to an object of class **SOMRReplicbl**.
ev	A pointer to the **Environment** 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.

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

        somrLock( somself, ev );
second  ...
method  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

receiver	A pointer to an object of class **SOMRReplicbl**.
ev	A pointer to the **Environment** 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

receiver	A pointer to an object of class **SOMRReplicbl**.
ev	A pointer to the **Environment** 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

receiver	A pointer to an object of class **SOMRReplicbl**.
ev	A pointer to the **Environment** 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

receiver A pointer to an object of class **SOMRReplicbl**.

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

clsname A character string representing the name of the class to be logged.

buffer A character buffer for logged information. (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.)

bufferlen An integer representing the size of the buffer required.

intObjId 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

somrRepInit Method

Purpose

Makes the object ready for replication.

IDL Syntax

long somrRepInit (
in char *lType***,**
in char *mode***);**

Description

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

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

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

Parameters

receiver A pointer to an object of class **SOMRReplicbl**.

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

lType A **char** indicating the type of logging used: **v** indicates *value logging*; **o** indicates *operation logging*.

mode A **char** indicating whether the object is open for reading (**r**) or writing (**w**).

Return Values

The **somrRepInit** method returns1 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 = _somrRepInit( 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: somrRepUninit

somrRepUninit Method

Purpose

Destroys the setup for replication.

IDL Syntax

void somrRepUninit ( );

Description

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

Parameters

receiver	A pointer to an object of class **SOMRReplicbl**.
ev	A pointer to the **Environment** structure for the caller.

Return Values

None.

Examples

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

Original Class

SOMRReplicbl

Related Information

Methods: somrRepInit

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

receiver A pointer to an object of class **SOMRReplicbl**.

ev A pointer to the **Environment** 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.

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

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

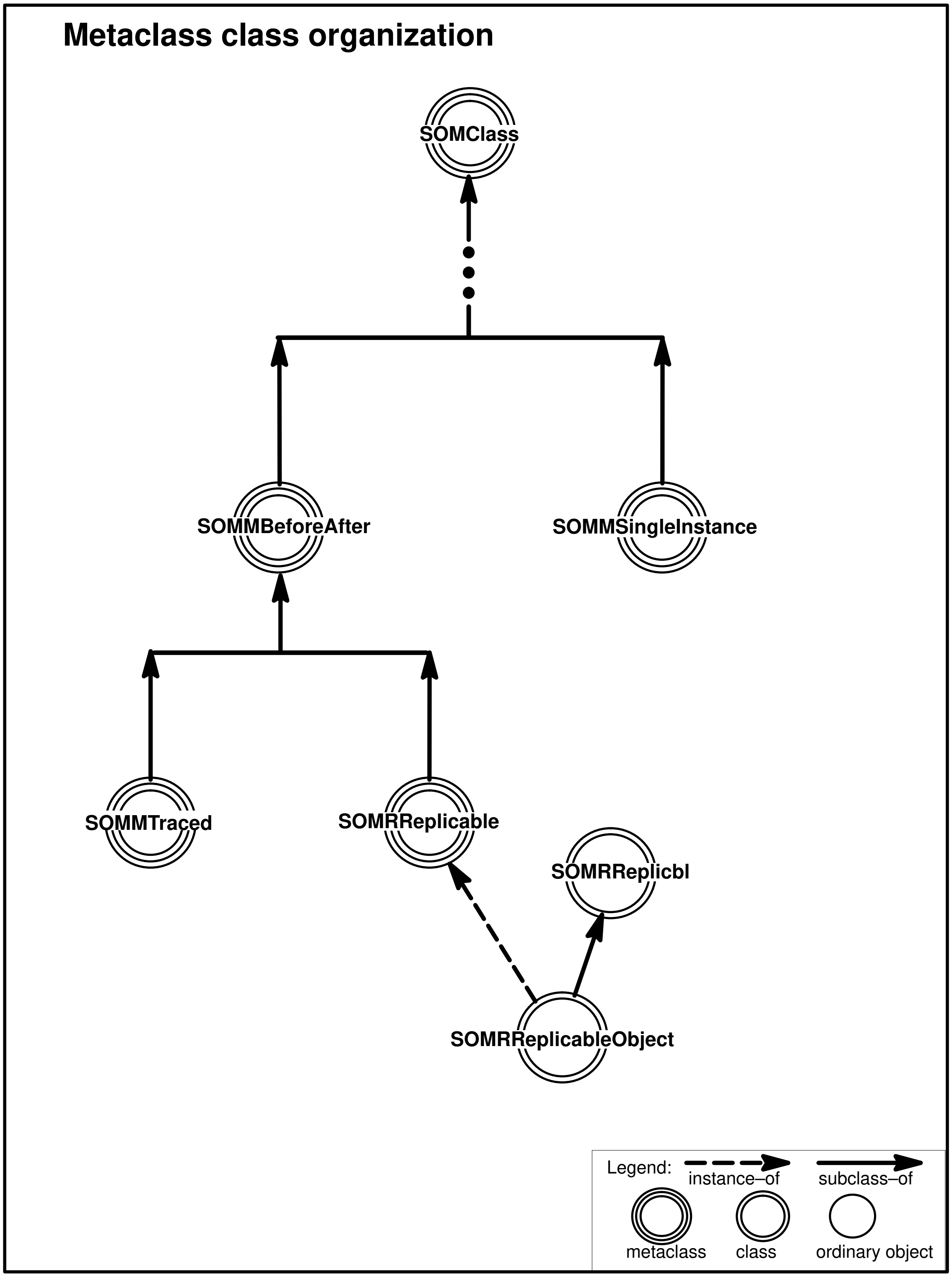
Original Class

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

sombacls

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.

receiver A pointer to an object (class) of metaclass **SOMMBeforeAfter** representing the class object that supports the method (such as, "myMethod") for which the "after" method will apply.

ev A pointer where the method can return exception information if an error is encountered. The dispatch method of **SOMMBeforeAfter** sets this parameter to NULL before dispatching the first **sommBeforeMethod**.

object A pointer to the instance of the receiver on which the method is invoked.

methodId The SOM ID of the method (such as, "myMethod") that was invoked.

returnedvalue A pointer to the value returned by invoking the method ("myMethod") on an object.

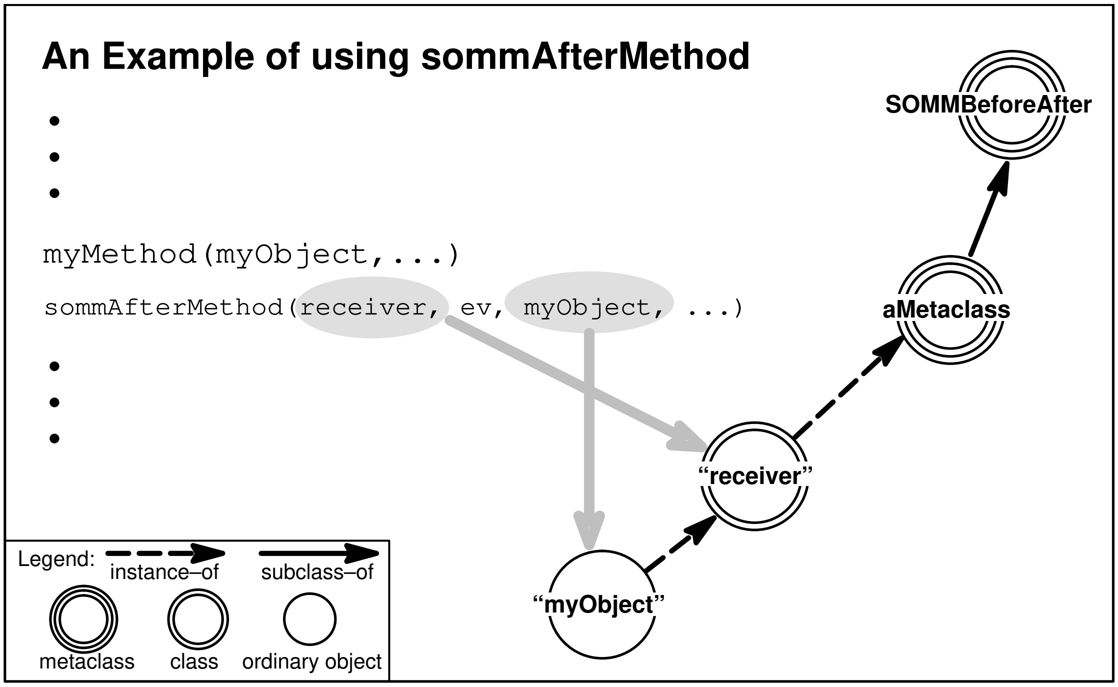
ap The list of input arguments to the method ("myMethod").

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.

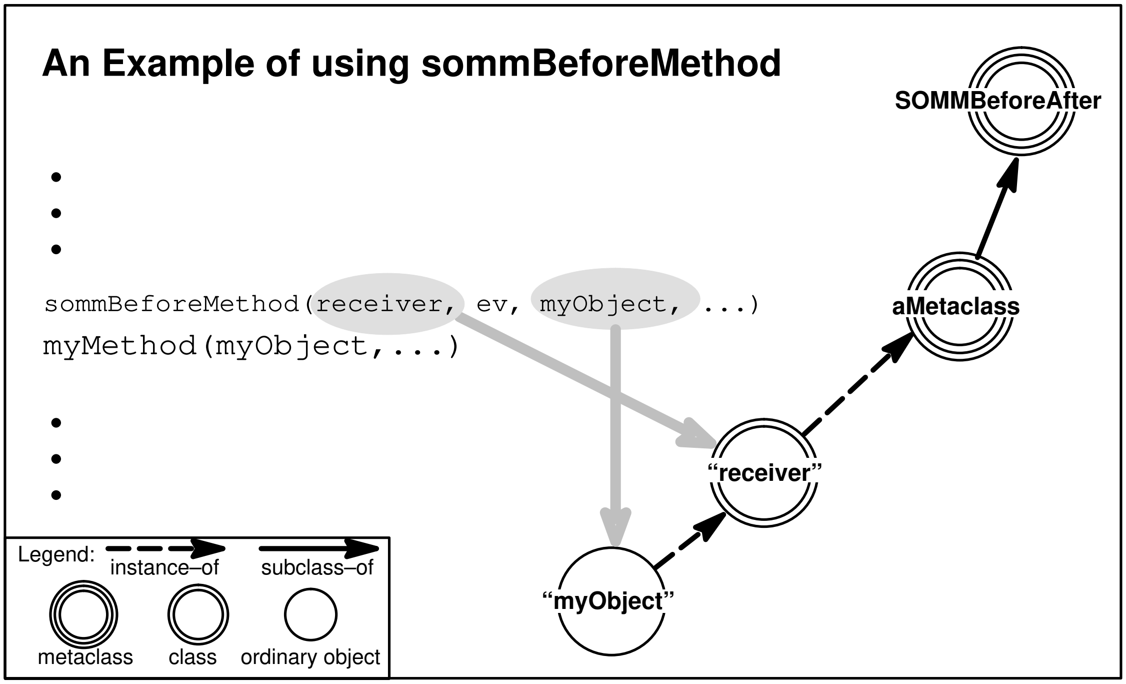
receiver	A pointer to an object (class) of metaclass **SOMMBeforeAfter** representing the class object that supports the method (such as, "myMethod") for which the "before" method will apply.
ev	A pointer where the method can return exception information if an error is encountered. The dispatch method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



An Example of using sommBeforeMethod

```
•
•
•
sommBeforeMethod(receiver, ev, myObject, ...)
myMethod(myObject,...)

•
•
•
```

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

somInit
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

receiver	A pointer to a class object whose metaclass is **SOMMSingleInstance** (or is a subclass of it).
ev	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 sommTraceIsOn

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 **somrRepInit** on instances of **SOMRReplicableObject**.

File Stem

somrcls

Base Class

SOMRReplicbl

Metaclass

SOMRReplicable

Ancestor Classes

SOMRReplicbl, **SOMRNameable**, **SOMRLinearizable**, **SOMObject**

New Methods

somrLoggingType
somrReplicableExemptMethod

Overriding Methods

somrRepInit
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

receiver	A pointer to an object that is a **SOMRReplicableObject**.
ev	A pointer to the **Environment** structure for the calling method.

Return Value

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

Example

```
RepObject x;
x = RepObjectNew();
_somrSetObjName(x,ev,”aRepObject”);
_somrRepInit(x,ev,’o’,’w’);
if ( ’o’ == _somrLoggingType(x,ev) )
        somPrintf( ”This will print” );
```

Original Class

SOMRReplicableObject

Related Information

Methods: somrRepInit

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

receiver	A pointer to an object that is a **SOMRReplicableObject**.
ev	A pointer to the **Environment** structure for the calling method.
methodId	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.

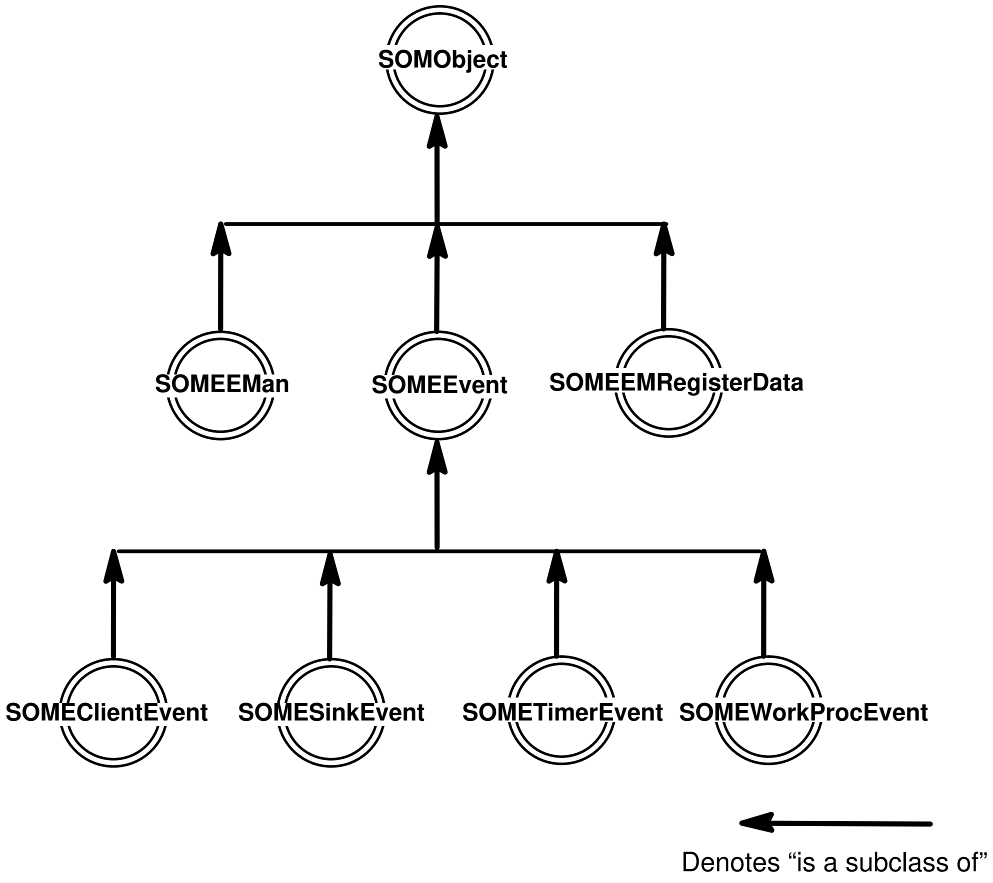
```
boolean somrReplicableExemptMethod( somId methodId ) {
        if ( somCompareIds( methodId, somIdFromString(”foo”) ) )
                return TRUE;
        else
                return FALSE;
}
```

Original Class

SOMRReplicableObject

Event Management Framework Reference

Event Management Framework Class Organization



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

somInit

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

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

Return Value

A pointer to user-defined client event data.

Original Class

SOMEClientEvent

Related Information

Methods: somevSetEventClientData

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

receiver	A pointer to an object of class **SOMEClientEvent.**
ev	A pointer to the **Environment** 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

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

Return Value

None.

Original Class

SOMEClientEvent

Related Information

Methods: somevGetEventClientData

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

receiver	A pointer to an object of class **SOMEClientEvent.**
ev	A pointer to the **Environment** structure for the calling method.
clientType	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 **someProcessEvents** 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 **someProcessEvents** call is preferable.

<u>AIX Specifics</u>:
On AIX this event manager supports Timer, Sink (any file, pipe, socket, or Message Queue), Client and WorkProc events.

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

<u>Thread Safety</u>:
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

someProcessEvents
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

receiver	A pointer to an object of class **SOMEEMan.**
ev	A pointer to the **Environment** structure for the calling method.
registrationId	The registration ID of the event interest whose data is being changed.
registerData	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

receiver	A pointer to an object of class **SOMEEMan.**
ev	A pointer to the **Environment** 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

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

someProcessEvents Method

Purpose

Processes infinite events.

IDL Syntax

void someProcessEvents ( );

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

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

Return Value

None.

Example

```
#include <eman.h>

main()
{
Environment *testEnv = somGetGlobalEnvironment();
SOMEEMan *some_gEMan = SOMEEManNew();
 /* Do some registrations */
    ...
_someProcessEvents(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

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

receiver A pointer to an object of class **SOMEEMan.**

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

registerData 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.

targetObject A pointer to the object that is the target of the callback method.

targetMethod The name of the callback method.

targetData 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

receiver A pointer to an object of class **SOMEEMan**.

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

registerData A pointer to registration data object that contains all the necessary information about the event for which an interest is being registered with EMan.

targetObject A pointer to the object which is the target of the callback method

callbackEv A pointer to the Environment structure to be passed to the callback method

targetMethod The name of the callback method.

targetData 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 );
regId1 = _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

receiver A pointer to an object of class **SOMEEMan.**

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

registerData A pointer to registration data object that contains all the necessary information about the event for which an interest is being registered with EMan.

targetProcedure
A pointer to the procedure (callback) that is called when the registered event occurs.

targetData 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

receiver	A pointer to an object of class **SOMEEMan.**
ev	A pointer to the **Environment** 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

receiver	A pointer to an object of class **SOMEEMan.**
ev	A pointer to the **Environment** 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

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

receiver	A pointer to an object of class **SOMEEMRegisterData.**
ev	A pointer to the **Environment** 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

receiver A pointer to an object of class **SOMEEMRegisterData.**

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

clientType 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

receiver	A pointer to an object of class **SOMEEMRegisterData.**
ev	A pointer to the **Environment** structure for the calling method.
eventType	A bit constant indicating the type of event being registered with EMan.
ap	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

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

receiver A pointer to an object of class **SOMEEMRegisterData.**

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

sinkmask 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

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

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

somInit

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

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

Return Value

An event timestamp in milliseconds.

Original Class

SOMEEvent

Related Information

Methods: somevSetEventTime

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

receiver	A pointer to an object of class **SOMEEvent.**
ev	A pointer to the **Environment** 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

receiver	A pointer to an object of class **SOMEEvent.**
ev	A pointer to the **Environment** structure for the calling method.
time	The time of day expressed in milliseconds.

Return Value

None.

Original Class

SOMEEvent

Related Information

Methods: somevGetEventTime

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

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

somInit

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

receiver	A pointer to an object of class **SOMESinkEvent.**
ev	A pointer to the **Environment** 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

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

SOMETimerEvent 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

somInit

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

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

Return Value

The interval time in milliseconds.

Original Class

SOMETimerEvent

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

receiver	A pointer to an object of class **SOMETimerEvent.**
ev	A pointer to the **Environment** structure for the calling method.
interval	The timer interval in milliseconds.

Return Value

None.

Original Class

SOMETimerEvent

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

somInit

Index