SOMobjects Developer Toolkit

Programmers Reference Manual

Reference material for the classes, methods, functions, and macros provided by the System Object Model and its accompanying frameworks

Version 2.1
October 1994

Note: Before using this information and the product it supports, be sure to read the trademark information under “Trademarks” on page xv.

Version 2.1 (October 1994)

SOMobjects Toolkit Programmers Reference Manual

Contents

About This Book

This book gives reference material for the **System Object Model** (**SOM**) of the **SOMobjects Developer Toolkit.** In particular, it contains a reference page for every class, method, function, and macro provided by the SOM run-time library, the DSOM run-time library, the Persistence Framework, the Replication Framework, the Interface Repository Framework, and the Event Management Framework. It also includes documentation of the utility metaclasses provided by the SOMobjects Developer Toolkit, and each of their methods.

In addition to this book, refer to the *SOMobjects Developer Toolkit Users Guide* for introductory information. Also, refer to the *Emitter Framework Guide and Reference* for documentation of the Emitter Framework of the SOMobjects Toolkit, and to the *SOMobjects Developer Toolkit Collection Classes Reference Manual* for documentation of the collection classes and methods.

How This Book Is Organized

At the highest level, this book is organized by framework. Within each framework, the reference pages describe the classes in alphabetical order, with the methods of each class given in alphabetical order following their corresponding class. Similarly, related functions and SOM macros are given in separate alphabetical sequences in the corresponding section. The reference page for a SOM **class** contains the following topics:

Description:	A description of the class.
File Stem:	The file stem for the class's IDL interface specification (.idl) file and its usage binding (.h/.xh) files.
Base Class:	The class's direct base (parent) classes.
Ancestor Classes:	The class's ancestor (indirect base) classes.
Metaclass:	The class's metaclass.
New Methods:	The names of the methods that the class introduces (grouped roughly according to purpose). Each new method is documented on a separate reference page.
Overriding Methods:	The names of the methods that the class overrides from ancestor classes

The reference page for a **method** of a SOM class contains the following topics:

Purpose:	The purpose of the method in brief.
Syntax:	The method's C/C++ procedure prototype (which includes the method procedure's return type and the names and types of its parameters). The in/out/inout keywords associated with each of the method's parameters in the method's IDL declaration are also shown. These keywords are shown for information only; they are not actually present in the method procedure prototype.
Description:	A description of the method's use.
Parameters:	A description of each of the method procedure's parameters**.**
Return Value:	A description of the method's return value.
Example:	An example of using or overriding the method, if available. Although methods of SOM classes are language neutral (i.e., they can be invoked from any programming language that can use SOM), the examples given here are written in C.
Original Class:	The name of the class that introduces the method (the class is documented separately in this book)**.**
Related Information:	Related methods and functions (and macros, for the SOM kernel) that can be found in this book.

The reference page for a **function** has the following topics:

Purpose:	The purpose of the function in brief.
Syntax:	The function's prototype (which includes the return type and the names and types of the parameters).
Description:	A description of the function's use.
Parameters:	A description of each of the function's parameters.
Return Value:	A description of the function's return value.
Example:	An example of using the function, if available.
Related Information:	Related methods and functions (and macros, for the SOM kernel) that can be found in this book.

The reference page for a **macro** has the following topics:

Purpose:	The purpose of the macro in brief.
Syntax:	The syntax for invoking the macro.
Description:	A description of the macro's use.
Parameters:	A description of each of the macro's parameters.
Expansion:	A description of the macro's expansion (although the exact code expansion is not always given).
Example:	An example of invoking the macro, if available.
Related Information:	Related macros and functions that can be found in this book.

Who Should Use This Book

This book is for the professional programmer using the SOMobjects Developer Toolkit to build object-oriented class libraries or application programs that use SOM class libraries or the frameworks in the SOMobjects Developer Toolkit.

This book assumes that you are an experienced programmer and that you have a general familiarity with the basic notions of object-oriented programming. Practical experience using an object-oriented programming language is helpful, but not essential.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
IBM
Operating System/2
OS/2
OS/2 Workplace Shell
RISC System 6000
SOMobjects
System Object Model

For convenience, the acronym "SOM" is used in this publication to reference the technology of the System Object Model, and the term "SOM Compiler" is used to reference the compiler of the System Object Model.

Each of the following terms used in this publication is a trademark of another company:

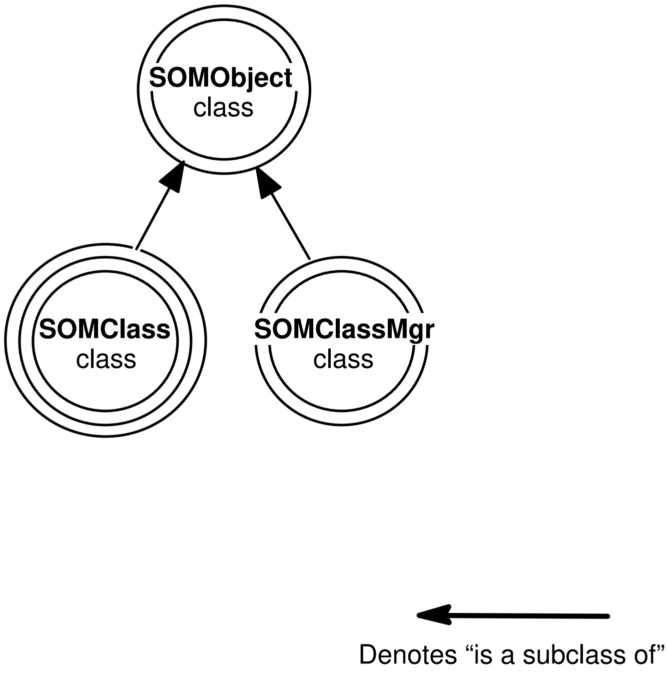
Intel	Intel Corporation
IPX	Novell Corporation
Lotus 1-2-3	Lotus Development Corporation
Microsoft EXCEL	Microsoft Corporation
Microsoft Windows	Microsoft Corporation
NetWare	Novell Corporation
Objective-C	The Stepstone Corporation
Smalltalk	Digitalk Inc.

The term "ANSI C" used throughout this publication refers to American National Standard X3.159–1989.

The term "CORBA" used throughout this publication refers to the Common Object Request Broker Architecture standards promulgated by the Object Management Group, Inc.

SOM Kernel Reference

SOM Kernel Class Organization



somApply Function

Purpose

Invokes an apply stub. Apply stubs are never invoked directly by SOM users, the **somApply** function must be used instead.

Syntax

boolean somApply (
SOMObject objPtr,
somToken *retVal,
somMethodDataPtr mdPtr,
va_list args);

Description

somApply provides a single uniform interface through which it is possible to call any method procedure. The interface is based on the caller passing: the object to which the method procedure is to be applied; a return address for the method result; a *somMethodDataPtr* indicating the desired method procedure; and an ANSI standard **va_list** structure containing the method procedure arguments. Different method procedures expect different argument types and return different result types, so the purpose of **somApply** is to select an *apply stub* appropriate for the specific method involved, according to the supplied method data, and then call this apply stub. The apply stub removes the arguments from the **va_list**, calls the method procedure with these arguments, accepts the returned result, and then copies this result to the location pointed to by *retVal*.

The method procedure used by the apply stub is determined by the content of the **somMethodData** structure pointed to by *mdPtr*. The class methods **somGetMethodData** and **somGetNthMethodData** are used to load a **somMethodData** structure. These methods resolve static method procedures based on the receiving class's instance method table.

The SOM API requires that information necessary for selecting an apply stub be provided when a new method is registered with its introducing class (via the methods **somAddStaticMethod** or **somAddDynamicMethod**). This is required because SOM itself needs apply stubs when dispatch method resolution is used. C and C++ implementation bindings for SOM classes support this requirement, but SOM does not terminate execution if this requirement is not met by a class implementor. Thus, it is possible that there may be methods for which **somApply** cannot select an appropriate apply stub. The **somMethodData** structure for the method can be inspected before calling **somApply** to verify that the method data contains sufficient information to select an appropriate apply stub: either the *applyStub* component or the *stubInfo* component of this structure must be non-NULL. If these conditions are met, then **somApply** performs as described above, and a TRUE value is returned; otherwise FALSE is returned.

Parameters

objPtr
A pointer to the object on which the method procedure is to be invoked.

retVal
A pointer to the memory region into which the result returned by the method procedure is to be copied. This pointer **cannot** be null (even in the case of method procedures whose returned result is void).

mdPtr
A pointer to the **somMethodData** structure that describes the method whose procedure is to be executed by the apply stub.

args
A pointer to a memory region in which all of the arguments to the method procedure have been laid out in consecutive addresses, according to the protocol implemented by **va_lists**. The first entry of the **va_list must** be *objPtr*.

Furthermore, all arguments on the **va_list** must appear in widened form, as defined by ANSI C. For example, **floats** must appear as **doubles**, and **chars** and **shorts** must appear as **ints**.

Return Value

The **somApply** function returns 1 (TRUE) if it executes successfully, or 0 (FALSE) otherwise.

C++ Example

```
#include <somcls.xh>
#include <string.h>
#include <stdarg.h>
main()
{   va_list args = (va_list) SOMMalloc(4);
    va_list push = args;
    string result;
    SOMClass *scObj;
    somMethodData md;

    somEnvironmentNew(); /* Init environment */
    scObj = _SOMClass;   /* The SOMClass object */

    scObj->somGetMethodData(somIdFromString(”somGetName”), &md);
    va_arg(push, SOMClass*) = scObj;

    somApply(scObj, (somToken*)&result, &md, args);
    SOM_Assert(!strcmp(result,”SOMClass”), SOM_Fatal);
    /* result is ”SOMClass” */
}
```

Related Information

Data Structures: SOMObject (somobj.idl), **somMethodData** (somapi.h), **somToken** (sombtype.h), **somMethodPtr** (sombtype.h), **va_list** (stdarg.h)

Methods: somGetMethodData, somGetNthMethodData, somAddDynamicMethod(somcls.idl)

somBeginPersistentIds Function

Purpose

Tells SOM to begin a “persistent ID interval.”

Syntax

void somBeginPersistentIds ( );

Description

The **somBeginPersistentIds** function informs the SOM ID manager that strings for any new SOM IDs that are registered will not be freed or modified. This allows the ID manager to use a pointer to the string in the unregistered ID as the master copy of the ID’s string, rather than making a copy of the string. This makes ID handling more efficient.

Parameters

None.

Return Value

None.

C Example

```
#include <som.h>
/* This is the way to create somIds efficiently */
static string id1Name = "whoami";
static somId somId_id1 = &id1Name;
/*
   somId_id1 will be registered the first time it is used
   in an operation that takes a somId, or it can be explicitly
   registered using somCheckId.
*/

main()
{
   somId id1, id2;
   string id2Name = "whereami";

   somEnvironmentNew();
   somBeginPersistentIds();
   id1 = somCheckId(somId_id1); /* registers the id as persistent */
   somEndPersistentIds();
   id2 = somIdFromString(id2Name); /* registers the id */

   SOM_Assert(!strcmp("whoami", somStringFromId(id1)), SOM_Fatal);
   SOM_Assert(!strcmp("whereami", somStringFromId(id2)), SOM_Fatal);

   id1Name = "it does matter"; /* because it is persistent */
   id2Name = "it doesn't matter"; /* because it is not persistent */

   SOM_Assert(strcmp("whoami", somStringFromId(id1)), SOM_Fatal);
   /* The id1 string has changed */
   SOM_Assert(!strcmp("whereami", somStringFromId(id2)), SOM_Fatal);
   /* the id2 string has not */
}
```

Related Information

Functions: somCheckId, somRegisterId, somIdFromString, somStringFromId, somCompareIds, somTotalRegIds, somUniqueKey, somSetExpectedIds, somEndPersistentIds

somBuildClass Function

Purpose

Automates the process of building a new SOM class object.

Syntax

```
SOMClass  somBuildClass (
                        unsigned long inheritVars,
                        somStaticClassInfoPtr sciPtr,
                        long majorVersion,
                        long minorVersion);
```

Description

The **somBuildClass** function accepts declarative information defining a new class that is be built, and performs the activities required to build and register a correctly functioning class object. The C and C++ implementation bindings use this function to create class objects.

Parameters

inheritVars	A bit mask that determines inheritance from parent classes. A mask containing all ones is an appropriate default.
sciPtr	A pointer to a structure holding static class information.
majorVersion	The major version number for the class.
minorVersion	The minor version number for the class.

Example

See any .ih or .xih implementation binding file for details on construction of the required data structures.

Return Value

The **somBuildClass** function returns a pointer to a class object.

Related Information

Data Structures: somStaticClassInfo (somapi.h)

somCheckId Function

Purpose

Registers a SOM ID.

Syntax

somId somCheckId (somId *id*);

Description

The **somCheckId** function registers a SOM ID and converts it into an internal representation. The input SOM ID is returned. If the ID is already registered, this function has no effect.

Parameters

id	The **somId** to be registered.

Return Value

The registered **somId.**

Example

See function **somBeginPersistentIds**.

Related Information

Data Structures: somId (sombtype.h)

Functions: somRegisterId, somIdFromString, somStringFromId, somCompareIds, somTotalRegIds, somSetExpectedIds, somUniqueKey, somBeginPersistentIds, somEndPersistentIds

somClassResolve Function

Purpose

Obtains a pointer to the procedure that implements a static method for instances of a particular SOM class.

Syntax

somMethodPtr somClassResolve (SOMClass *cls,* **somMToken** *mToken***);**

Description

The **somClassResolve** function is used to obtain a pointer to the procedure that implements the specified method for instances of the specified SOM class. The returned procedure pointer can then be used to invoke the method. The **somClassResolve** function is used to support "casted" method calls, in which a method is resolved with respect to a specified class rather than the class of which an object is a direct instance. The **somClassResolve** function can only be used to obtain a method procedure for a static method (a method declared in an IDL specification for a class); dynamic methods do not have method tokens.

The SOM language usage bindings for C and C++ do not support casted method calls, so this function must be used directly to achieve this functionality. Whenever using SOM method procedure pointers, it is necessary to indicate the use of system linkage to the compiler. The way this is done depends on the compiler and the system being used. However, C and C++ usage bindings provide an appropriate typedef for this purpose. The name of the typedef is based on the name of the class that introduces the method, as illustrated in the example below.

Parameters

cls — A pointer to the class object whose instance method procedure is required.

mToken — The method token for the method to be resolved. The SOM API requires that if the class "XYZ" introduces the static method "foo", then the method token for "foo" is found in the class data structure for "XYZ" (called XYZClassData) in the structure member named "foo" (that is, at XYZClassData.foo). Method tokens can also be obtained using the **somGetMethodToken** method.

Return Value

A **somMethodPtr** pointer to the procedure that implements the specified method for the specified class of SOM object.

C++ Example

```
// SOM IDL for class A and class B
#include <somobj.idl>
module scrExample {
   interface A : SOMObject { void foo(); implementation {
                             callstyle=oidl; }; };
   interface B : A  {  implementation { foo: override; }; };
};

// Example C++ program to implement and test module scrExample
#define SOM_Module_screxample_Source
#include <scrExample.xih>
#include <stdio.h>

SOM_Scope void SOMLINK scrExample_Afoo(scrExample_A *somSelf);
{  printf("1\n"); }

SOM_Scope void SOMLINK scrExample_Bfoo(scrExample_B *somSelf);
{ printf("2\n"); }

main()
{
   scrExample_B  *objPtr = new scrExample_B;

   // This prints 2
   objPtr->foo();

   // This prints 1
   ((somTD_scrExample_A_foo) /* A necessary method procedure cast */
      somClassResolve(
          _scrExample_A, // the A class object
          scrExample_AClassData.foo) // the foo method token
      ) /* end of method procedure expression */
      (objPtr); /* method arguments */

   // This prints 2
   ((somTD_scrExample_A_foo) /* A necessary method procedure cast */
      somClassResolve(
          _scrExample_B, // the B class object
          scrExample_AClassData.foo) // the foo method token
      ) /* end of method procedure expression */
      (objPtr); /* method arguments */

}
```

Related Information

Data Structures: somMethodPtr (sombtype.h), **SOMClass** (somcls.idl), **somMToken** (somapi.h)

Functions: somResolveByName, somParentResolve, somParentNumResolve, somResolve

Methods: somDispatch, somClassDispatch, somFindMethod, somFindMethodOk, somGetMethodToken

Macros: SOM_Resolve, SOM_ResolveNoCheck

somCompareIds Function

Purpose

Determines whether two SOM IDs represent the same string.

Syntax

int somCompareIds (somId *id1,* **somId** *id2***);**

Description

The **somCompareIds** function returns 1 if the two input IDs represent strings that are equal; otherwise, it returns 0.

Parameters

id1 The first SOM ID to be compared.

id2 The second SOM ID to be compared.

Return Value

Returns returns 1 if the two input IDs represent strings that are equal; otherwise, it returns 0.

C Example

```
#include <som.h>
main()
{
   somId id1, id2, id3;

   somEnvironmentNew();
   id1 = somIdFromString("this");
   id2 = somIdFromString("that");
   id3 = somIdFromString("this");

   SOM_Test(somCompareIds(id1, id3));
   SOM_Test(! somCompareIds(id1, id2));
}
```

Related Information

Data Structures: somId (sombtype.h)

Functions: somCheckId, somRegisterId, somIdFromString, somStringFromId, somTotalRegIds, somSetExpectedIds, somUniqueKey, somBeginPersistentIds, somEndPersistentIds

somDataResolve Function

Purpose

Accesses instance data within an object.

Syntax

somToken somDataResolve (SOMObject *obj,* **somDToken** *dToken***);**

Description

The **somDataResolve** function is used to access instance data within an object. This function is of use primarily to class implementors (rather than class clients) who are not using the SOM C or C++ language bindings.

For C or C++ programmers with access to the C or C++ implementation bindings for a class, instance data can be accessed using the <className>GetData macro (which expands to a usage of **somDataResolve**).

Parameters

obj	A pointer to the object whose instance data is required.
dToken	A data token for the required instance data. The SOM API specifies that the data token for accessing the instance data introduced by a class is found in the *instanceDataToken* component of the auxiliary class data structure for that class. The example below illustrates this.

Return Value

A **somToken** (that is, a pointer) that points to the data in *obj* identified by the *dToken*.

C Example

The following C/C++ expression evaluates to the address of the instance data introduced by class “XYZ” within the object “obj”. This assumes that “obj” points to an instance of “XYZ” or a subclass of “XYZ”.

```
include <som.h>
somDataResolve(obj, XYZCClassData.instanceDataToken)
```

Related Information

Data Structures: somToken (sombtype.h), **SOMObject** (somobj.idl),
somDToken (somapi.h)

somEndPersistentIds Function

Purpose

Tells SOM to end a “persistent ID interval.”

Syntax

void somEndPersistentIds ( );

Description

The **somEndPersistentIds** function informs the SOM ID manager that strings for any new SOM IDs that are registered might be freed or modified by the client program. Thus, the ID manager must make a copy of the strings.

Parameters

None.

Return Value

None.

Example

See function **somBeginPersistentIds**.

Related Information

Functions: somCheckId, somRegisterId, somIdFromString, somStringFromId, somCompareIds, somTotalRegIds, somUniqueKey, somSetExpectedIds, somBeginPersistentIds

somEnvironmentEnd Function

Purpose

Provides general cleanup for applications.

Syntax

void somEnvironmentEnd ( );

Description

The **somEnvironmentEnd** function is a general cleanup function that must be called by all Windows applications before exiting. AIX and OS/2 programs may also invoke this function, but it is not required on these systems because all necessary SOM cleanup is performed by the operating system during program termination.

A convenience macro, **SOM_MainProgram,** which usually appears at the beginning of each application, adds the **somEnvironmentEnd** function to the “atexit” list. If the “atexit” mechanism does not work reliably with your compiler, or if you know that your program bypasses the normal program termination sequence, you should insert an explicit call to **somEnvironmentEnd** at the point where your main program exits. (All main programs for Windows must begin either with the **SOM_MainProgram** macro or with a call to the **somMainProgram** function.)

Parameters

None.

Return Value

None.

Related Information

Macros: SOM_MainProgram

somEnvironmentNew Function

Purpose

Initializes the SOM runtime environment.

Syntax

SOMClassMgr somEnvironmentNew ( );

Description

The **somEnvironmentNew** function creates the four primitive SOM objects (*SOMObject, SOMClass, SOMClassMg*r, and *SOMClassMgrObject*) and initializes global variables used by the SOM run-time environment. This function must be called before using any other SOM functions or methods (with the exception of **somSetExpectedIds**). If the SOM run-time environment has already been initialized, calling this function has no harmful effect.

Although this function must be called before using other SOM functions or methods, it needn't always be called explicitly, because the <*className*>**New** macros, the <*className*>**Renew** macros, the **new** operator, and the <*className*>**NewClass** procedures defined by the SOM C and C++ language bindings call **somEnvironmentNew** if needed.

Parameters

None.

Return Value

A pointer to the single class manager object active at run time. This class manager can be referred by the global variable *SOMClassMgrObject*.

Example

```
somEnvironmentNew();
```

Related Information

Functions: somExceptionId, somExceptionValue, somSetException, somGetGlobalEnvironment

somExceptionFree Function

Purpose

Frees the memory held by the exception structure within an **Environment** structure.

Syntax

void somExceptionFree (Environment ***_ev_);**

Description

The **somExceptionFree** function frees the memory held by the exception structure within an **Environment** structure.

Parameters

_ev_	A pointer to the **Environment** whose exception information is to be freed.

Return Value

None.

Example

See function **somSetException**.

Related Information

Data Structures: Environment (somcorba.h)

Functions: somExceptionId, somExceptionValue, somSetException, somGetGlobalEnvironment, somdExceptionFree (DSOM function)

somExceptionId Function

Purpose

Gets the name of the exception contained in an **Environment** structure.

Syntax

string somExceptionId (Environment **ev*)**;**

Description

The **somExceptionId** function returns the name of the exception contained in the specified **Environment** structure.

Parameters

ev	A pointer to an **Environment** structure containing an exception.

Return Value

The **somExceptionId** function returns the name of the exception contained in the specified **Environment** structure, as a string.

Example

See function **somSetException**.

Related Information

Data Structures: string (somcorba.h), **Environment** (somcorba.h)

Functions: somExceptionValue, somExceptionFree, somSetException, somGetGlobalEnvironment

somExceptionValue Function

Purpose

Gets the value of the exception contained in an **Environment** structure.

Syntax

somToken somExceptionValue (Environment ****ev*);**

Description

The **somExceptionValue** function returns the value of the exception contained in the specified **Environment** structure.

Parameters

ev	A pointer to an **Environment** structure containing an exception.

Return Value

The **somExceptionValue** function returns a pointer to the value of the exception contained in the specified **Environment** structure.

Example

See function **somSetException**.

Related Information

Data Structures: somToken (sombtype.h), **Environment** (somcorba.h)

Functions: somExceptionId, somExceptionFree, somSetException, somGetGlobalEnvironment

somGetGlobalEnvironment Function

Purpose

Returns a pointer to the current global **Environment** structure.

Syntax

Environment *somGetGlobalEnvironment ( );

Description

The **somGetGlobalEnvironment** function returns a pointer to the current global **Environment** structure. This structure can be passed to methods that require an **(Environment *)** argument. The caller can determine if the called method has raised an exception by testing whether

```
ev->_major != NO_EXCEPTION
```

If an exception has been raised, the caller can retrieve the name and value of the exception using the **somExceptionId** and **somExceptionValue** functions.

Parameters

None.

Return Value

A pointer to the current global **Environment** structure.

Example

See function **somSetException**.

Related Information

Data Structures: Environment (somcorba.h)

Functions: somExceptionId, somExceptionValue, somExceptionFree, somSetException

somIdFromString Function

Purpose

Returns the SOM ID corresponding to a given text string.

Syntax

somId somIdFromString (string *aString***);**

Description

The **somIdFromString** function returns the SOM ID that corresponds to a given text string.

Ownership of the **somId** returned by **somIdFromString** passes to the caller, which has the responsibility to subsequently free the **somId** using **SOMFree**.

Parameters

aString	The string to be converted to a SOM ID.

Return Value

Returns the SOM ID corresponding to the given text string.

Example

See function **somBeginPersistentIds**.

Related Information

Data Structures: somId (sombtype.h), **string** (somcorba.h)

Functions: somCheckId, somRegisterId, somStringFromId, somCompareIds, somTotalRegIds, somSetExpectedIds, somUniqueKey, somBeginPersistentIds, somEndPersistentIds

somIsObj Function

Purpose

Failsafe routine to determine whether a pointer references a valid SOM object.

Syntax

boolean somIsObj (somToken *memPtr***);**

Description

The **somIsOb**j function returns 1 if its argument is a pointer to a valid SOM object, or returns 0 otherwise. The function handles address faults, and does extensive consistency checking to guarantee a correct result.

Parameters

memPtr A **somToken** (a pointer) to be checked.

Return Value

The **somIsObj** function returns 1 if *obj* is a pointer to a valid SOM object, and 0 otherwise.

C++ Example

```
#include <stdio.h>
#include <som.xh>

void example(void *memPtr)
{
   if (!somIsObj(memPtr))
     printf("memPtr is not a valid SOM object.\n");
   else
     printf("memPtr points to an object of class %s\n",
             ((SOMObject *)memPtr)->somGetClassName());
}
```

Related Information

Data Structures: boolean (somcorba.h), **somToken** (sombtype.h)

somLPrintf Function

Purpose

Prints a formatted string in the manner of the C printf function, at the specified indentation level.

Syntax

long somLPrintf (long *level,* **string** *fmt,* ...**);**

Description

The **somLPrintf** function prints a formatted string using SOMOutCharRoutine, in the same manner as the C printf function. The implementation of SOMOutCharRoutine determines the destination of the output, while the C printf function is always directed to stdout. (The default output destination for SOMOutCharRoutine is stdout also, but this can be modified by the user.) The output is prefixed at the indicated level, by preceding it with 2*level spaces.

Parameters

level	The level at which output is to be placed.
fmt	The format string to be output.
varargs	The values to be substituted into the format string.

Return Value

Returns the number of characters written.

C Example

```
#include <somobj.h>
somLPrintf(5, "The class name is %s.\n", _somGetClassName(obj));
```

Related Information

Data Structures: string (somcorba.h)

Functions: somVprintf, somPrefixLevel, somPrintf, SOMOutCharRoutine

somMainProgram Function

Purpose

Performs SOM initialization on behalf of a new program.

Syntax

SOMClassMgr *somMainProgram ( );

Description

The **somMainProgram** function informs SOM about the beginning of a new thread of execution (called a *task* on Windows). The SOM Kernel then performs any needed initialization, including the deferred execution of the **SOMInitModule** functions found in statically-loaded class libraries. The **somMainProgram** function must appear near the beginning of all Windows main programs, and may also be used in AIX or OS/2 programs. When used, it supersedes any need to call the **somEnvironmentNew** function.

A convenience macro, **SOM_MainProgram**, which combines the execution of the **somMainProgram** function with the scheduling of the **somEnvironmentEnd** function during normal program termination, is available for C and C++ programmers.

Parameters

None.

Return Value

A pointer to the **SOMClassMgr** object.

Related Information

Functions: somEnvironmentNew, somEnvironmentEnd

Macros: SOM_MainProgram, SOM_ClassLibrary

somParentNumResolve Function

Purpose

Obtains a pointer to a procedure that implements a method, given a list of method tables.

Syntax

somMethodPtr somParentNumResolve (
somMethodTabs *parentMtab,*
int *parentNum***,**
somMToken *mToken***);**

Description

The **somParentNumResolve** function is used to make parent method calls by the C and C++ language implementation bindings. The **somParentNumResolve** function returns a pointer to a procedure for performing the specified method. This pointer is selected from the specified method table, which is intended to be the method table corresponding to a parent class.

For C and C++ programmers, the implementation bindings for SOM classes provide convenient macros for making parent method calls (the "parent_" macros).

Parameters

parentMtab A list of method tables for the parents of the class being implemented. The SOM API specifies that the list of parent method tables for a given class be stored in the auxiliary class data structure of the class, in the *parentMtab* component. Thus, for the class "XYZ", the parent method table list is found in location *XYZCClassData.parentMtab.* Parent method table lists are available from class objects via the method call **somGetPClsMtabs**.

parentNum The position of the parent for which the method is to be resolved. The order of a class's parents is determined by the order in which they are specified in the interface statement for the class. (The first parent is number 1.)

mToken The method token for the method to be resolved. The SOM API requires that if the class "XYZ" introduces the static method "foo", then the method token for "foo" is found in the class data structure for "XYZ" (called XYZClassData) in the structure member named "foo" (that is, at XYZClassData.foo). Method tokens can also be obtained using the **somGetMethodToken** method.

Return Value

A **somMethodPtr** pointer to a procedure that implements the specified method, selected from the specified method table.

C++ Example

```
// SOM IDL for class A and class B
#include <somobj.idl>
module spnrExample {
   interface A : SOMObject  { void foo();  implementation {
                              callstyle=oidl; }; };
   interface B : A  {  implementation { foo: override; }; };
};

// Example C++ program to implement and test  module scrExample
#define SOM_Module_spnrexample_Source
#include <spnrExample.xih>
#include <stdio.h>

SOM_Scope void SOMLINK spnrExample_Afoo(spnrExample_A *somSelf);
{  printf("1\n"); }

SOM_Scope void SOMLINK spnrExample_Bfoo(spnrExample_B *somSelf);
{ printf("2\n"); }

main()
{
   spnrExample_B  *objPtr = new spnrExample_B;

   // This prints  2
   objPtr->foo();

   // This prints 1
   ((somTD_spnrExample_A_foo) /* This method procedure expression cast
                                 is necessary */
      somParentNumResolve(
           objPtr->somGetClass()->somGetPClsMtabs(),
           1,
           spnrExample_AClassData.foo) // the foo method token
      ) /* end of method procedure expression */
      (objPtr); /* method arguments */
}
```

Related Information

Data Structures: somMethodPtr (sombtype.h), **somMethodTabs** (somapi.h), **somMToken** (somapi.h)

Functions: somResolveByName, somResolve, somParentNumResolve, somClassResolve

Methods: somGetPClsMtab, somGetPClsMtabs, somGetMethodToken

Macros: SOM_ParentNumResolve, SOM_Resolve, SOM_ResolveNoCheck

somParentResolve Function

Purpose

Obtains a pointer to a procedure that implements a method, given a list of method tables. Obsolete but still supported.

Syntax

somMethodPtr somParentResolve (somMethodTabs *parentMtab,*
somMToken *mToken***);**

Description

The **somParentResolve** function is used by old, single-parent class binaries to make parent method calls. The function is obsolete, but is still supported. The **somParentResolve** function returns a pointer to the procedure that implements the specified method. This pointer is selected from the first method table in the parentMtab list.

Parameters

parentMtab A list of parent method tables, the first of which is the method table for the parent class for which the method is to be resolved. The SOM API specifies that the list of parent method tables for a given class be stored in the auxiliary class data structure of the class, in the *parentMtab* component. Thus, for the class “XYZ”, the parent method table list is found in location XYZCClassData.parentMtab. Parent method table lists are available from class objects via the method call **somGetPClsMtabs**.

mToken The method token for the method to be resolved. The SOM API requires that if the class “XYZ” introduces the static method “foo”, then the method token for “foo” is found in the class data structure for “XYZ” (called XYZClassData) in the structure member named “foo” (that is, at XYZClassData.foo). Method tokens can also be obtained using the **somGetMethodToken** method.

Return Value

A **somMethodPtr** pointer to the procedure that implements the specified method, selected from the first method table.

Related Information

Data Structures: somMethodPtr (sombtype.h), **somMethodTabs** (somapi.h), **somMToken** (somapi.h)

Functions: somResolveByName, somResolve, somParentNumResolve, somClassResolve

Methods: somDispatch, somClassDispatch, somFindMethod, somFindMethodOk, somGetMethodToken

Macros: SOM_Resolve, SOM_ResolveNoCheck

somPrefixLevel Function

Purpose

Outputs blanks to prefix a line at the indicated level.

Syntax

void somPrefixLevel (long *level***);**

Description

The **somPrefixLevel** function outputs blanks (via the **somPrintf** function) to prefix the next line of output at the indicated level. (The number of blanks produces is 2*level.) This function is useful when overriding the **somDumpSelfInt** method, which takes the level as an argument.

Parameters

level	The level at which the next line of output is to start.

Return Value

None.

C/C++ Example

```
#include <som.h>
somPrefixLevel(5);
```

Related Information

Functions: somPrintf, somVprintf, somLPrintf, SOMOutCharRoutine

somPrintf Function

Purpose

Prints a formatted string in the manner of the C printf function.

Syntax

long somPrintf (string *fmt,* ...**);**

Description

The **somPrintf** function prints a formatted string using function **SOMOutCharRoutine**, in the same manner as the C printf function. The implementation of **SOMOutCharRoutine** determines the destination of the output, while the C printf function is always directed to stdout. (The default output destination for **SOMOutCharRoutine** is stdout also, but this can be modified by the user.)

Parameters

fmt	The format string to be output.
varargs	The values to be substituted into the format string.

Return Value

Returns the number of characters written.

C Example

```
#include <somcls.h>
somPrintf(”The class name is %s.\n”, _somGetClassName(obj));
```

Related Information

Functions: somVprintf, somPrefixLevel, somLPrintf, SOMOutCharRoutine

somRegisterId Function

Purpose

Registers a SOM ID and determines whether or not it was previously registered.

Syntax

int somRegisterId (somId *id);*

Description

The **somRegisterId** function registers a SOM ID and converts it into an internal representation. If the ID is already registered, **somRegisterId** returns 0 and has no effect. Otherwise, **somRegisterId** returns 1.

Parameters

id The **somId** to be registered.

Return Value

If the ID is already registered, **somRegisterId** returns 0. Otherwise, s**omRegisterId** returns 1.

C Example

```
#include <som.h>
static string s = "unregistered";
static somId sid = &s;
main()
{
   somEnvironmentNew();
   SOM_Test(somRegisterId(sid) == 1);
   SOM_Test(somRegisterId(somIdFromString("registered")) == 0);
}
```

Related Information

Data Structures: somId (sombtype.h)

Functions: somCheckId, somIdFromString, somStringFromId, somCompareIds, somTotalRegIds, somSetExpectedIds, somUniqueKey, somBeginPersistentIds, somEndPersistentIds

somResolve Function

Purpose

Obtains a pointer to the procedure that implements a method for a particular SOM object.

Syntax

somMethodPtr somResolve (SOMObject *obj,* **somMToken** *mToken***);**

Description

The **somResolve** function returns a pointer to the procedure that implements the specified method for the specified SOM object. This pointer can then be used to invoke the method. The **somResolve** function can only be used to obtain a method procedure for a static method (one declared in an IDL or OIDL specification for a class); dynamic methods are not supported by method tokens.

For C and C++ programmers, the SOM usage bindings for SOM classes provide more convenient mechanisms for invoking methods. These bindings use the **SOM_Resolve** and **SOM_ResolveNoCheck** macros, which construct a method token expression from the class name and method name, and call **somResolve**.

Parameters

obj	A pointer to the object whose method procedure is required.
mToken	The method token for the method to be resolved. The SOM API requires that if the class "XYZ" introduces the static method "foo", then the method token for "foo" is found in the class data structure for "XYZ" (called XYZClassData) in the structure member named "foo" (that is, at XYZClassData.foo). Method tokens can also be obtained using the **somGetMethodToken** method.

Return Value

A **somMethodPtr** pointer to the procedure that implements the specified method for the specified SOM object.

C Example

```
// SOM IDL for class A and class B
#include <somobj.idl>
module srExample {
   interface A : SOMObject  { void foo();  implementation {
                              callstyle=oidl; }; };
   interface B : A  {  implementation { foo: override; }; };
};

// Example C++ program to implement and test  module scrExample
#define SOM_Module_srexample_Source
#include <srExample.ih>
#include <stdio.h>

SOM_Scope void SOMLINK srExample_Afoo(srExample_A *somSelf);
{  printf("1\n"); }

SOM_Scope void SOMLINK srExample_Bfoo(srExample_B *somSelf);
{ printf("2\n"); }

main()
{
   srExample_B  objPtr = srExample_BNew();

   /* This prints  2 */
   ((somTD_srExample_A_foo)   /* this method procedure expression cast
                                 is necessary */
       somResolve(objPtr, srExample_AClassData.foo)
       )     /* end of method procedure expression */
    (objPtr);
}
```

Related Information

Data Structures: somMethodPtr (sombtype.h), **somMToken** (somapi.h)

Functions: somResolveByName, somParentResolve, somParentNumResolve, somClassResolve

Methods: somDispatch, somClassDispatch, somFindMethod, somFindMethodOk, somGetMethodToken

Macros: SOM_Resolve, SOM_ResolveNoCheck

somResolveByName Function

Purpose

Obtains a pointer to the procedure that implements a method for a particular SOM object.

Syntax

somMethodPtr somResolveByName (SOMObject *obj,* **string** *methodName***);**

Description

The **somResolveByName** function is used to obtain a pointer to the procedure that implements the specified method for the specified SOM object. The returned procedure pointer can then be used to invoke the method. The C and C++ usage bindings use this function to support name-lookup methods.

This function can be used for invoking dynamic methods. However, the C and C++ usage bindings for SOM classes do not support dynamic methods, thus typedefs necessary for the use of dynamic methods are not available as with static methods. The function **somApply** provides an alternative mechanism for invoking dynamic methods that avoids the need for casting procedure pointers.

Parameters

obj	A pointer to the object whose method procedure is required.
methodName	A character string representing the name of the method to be resolved.

Return Value

A **somMethodPtr** pointer to the procedure that implements the specified method for the specified SOM object.

C Example

Assuming the static method "setSound," is introduced by the class "Animal", the following example will correctly invoke this method on an instance of "Animal" or one of its descendent classes.

```
#include <animal.h>
example(Animal myAnimal)
{
somTD_Animal_setSound
   setSoundProc = somResolveByName(myAnimal, "setSound");
setSoundProc(myAnimal, "Roar!");
}
```

Related Information

Data Structures: somMethodPtr (sombtype.h), **SOMObject** (somobj.idl), **string** (somcorba.h)

Functions: somResolve, somParentResolve, somParentNumResolve, somClassResolve

Methods: somDispatch, somClassDispatch, somFindMethod, somFindMethodOk

Macros: SOM_Resolve, SOM_ResolveNoCheck

somSetException Function

Purpose

Sets an exception value in an **Environment** structure.

Syntax

void somSetException (Environment ***ev,**
enum exception_type *major,*
string *exceptionName,*
somToken *params***);**

Description

The **somSetException** function sets an exception value in an **Environment** structure.

Parameters

ev — A pointer to the **Environment** structure in which to set the exception. This value must be either NULL or a value formerly obtained from the function **somGetGlobalEnvironment**.

major — An integer representing the type of exception to set.

exceptionName — The qualified name of the exception to set. The SOM Compiler defines, in the header files it generates for an interface, a constant whose value is the qualified name of each exception defined within the interface. This constant has the name “ex_<*exceptionName*>”, where <*exceptionName*> is the qualified (scoped) exception name. Where unambiguous, the usage bindings also define the short form “ex_<exceptionName>”, where <exceptionName> is unqualified.

params — A pointer to an initialized exception structure value. No copy is made of this structure; hence, the caller cannot free it. The **somExceptionFree** function should be used to free the **Environment** structure that contains it.

Return Value

None.

C Example

```
/* IDL declaration of class X:  */
   interface X : SOMObject {
      exception OUCH {long code1; long code2; };
      void foo(in long arg) raises (OUCH);
   };

/* implementation of foo method */
SOM_Scope void SOMLINK foo(X somSelf, Environment *ev, long arg)
{
   X_OUCH *exception_params; /* X_OUCH struct is defined
                                       in X's usage bindings    */

   if (arg > 5) /* then this is a very bad error */
   {
      exception_params = (X_OUCH*)SOM_Malloc(sizeof(X_OUCH));
      exception_params->code1 = arg;
      exception_params->code2 = arg-5;
      somSetException(ev, USER_EXCEPTION, ex_X_OUCH,
                          exception_params);
      /* the Environment ev now contains an X_OUCH exception, with
       * the specified exception_params struct. The constant
       * ex_X_OUCH is defined in foo.h. Note that exception_params
       * must be malloced.
       */
      return;
    }
...
}

main()
{
   Environment *ev;
   X x;

   somEnvironmentNew();
   x = Xnew();
   ev = somGetGlobalEnvironment();
   X_foo(x, ev, 23);
   if (ev->_major != NO_EXCEPTION) {
        printf("foo exception = %s\n", somExceptionId(ev));
        printf("code1 = %d\n",
                ((X_OUCH*)somExceptionValue(ev))->code1);
        /* finished handling exception. */
        /* free the copied id and the original X_OUCH structure: */
        somExceptionFree(ev);
   }
...
}
```

Related Information

Data Structures: Environment, exception_type, string (somcorba.h)

Functions: somExceptionId, somExceptionValue, somExceptionFree, somGetGlobalEnvironment

somSetExpectedIds Function

Purpose

Tells SOM how many unique SOM IDs a client program expects to use.

Syntax

void somSetExpectedIds (**unsigned long** *numIds);*

Description

The **somSetExpectedIds** function informs the SOM run-time environment how many unique SOM IDs a client program expects to use during its execution. This has the potential of slightly improving the program's space and time efficiency, if the value specified is accurate. This function, if used, must be called prior to any explicit or implicit invocation of the **somEnvironmentNew** function to have any effect.

Parameters

numIds	The number of SOM IDs the client program expects to use.

Return Value

None.

C Example

```
#include <som.h>
somSetExpectedIds(1000);
```

Related Information

Functions: somCheckId, somRegisterId, somIdFromString, somStringFromId, somCompareIds, somTotalRegIds, somUniqueKey, somBeginPersistentIds, somEndPersistentIds

somSetOutChar Function

Purpose

Changes the behavior of the **somPrintf** function.

Syntax

void somSetOutChar (
somTD_SOMOutCharRoutine * *outCharRtn***);**

Description

The **somSetOutChar** function is called to change the output character routine that **somPrintf** invokes. By default, **somPrintf** invokes a character output routine that goes to "stdout."

The execution of **somSetOutChar** affects only the application (or thread) in which it occurs. Thus, **somSetOutChar** is normally preferred over **SOMOutCharRoutine** for changing the output routine called by **somPrintf**, since **SOMOutCharRoutine** remains in effect for subsequent threads as well.

Some additional samples of **somSetOutChar** can be found in the "somapi.h" header file.

Parameters

outCharRtn A pointer to your routine that outputs a character in the way you want.

Return Value

None.

Example

```
#include <som.h>
static int irOutChar(char c);

static int irOutChar(char c)
{
    (Customized code goes here)
}

main (...)
{
    ...
    somSetOutChar((somTD_SOMOutCharRoutine *) irOutChar);
}
```

Related Information

Functions: somPrintf, SOMOutCharRoutine

somStringFromId Function

Purpose

Returns the string that a SOM ID represents.

Syntax

string somStringFromId (somId *id***);**

Description

The **somStringFromId** function returns the string that a given SOM ID represents.

Parameters

id	The SOM ID for which the corresponding string is needed.

Return Value

Returns the string that the given SOM ID represents.

Example

See function **somBeginPersistentIds**.

Related Information

Data Structures: string (somcorba.h), **somId** (sombtype.h)

Functions: somCheckId, somRegisterId, somIdFromString, somCompareIds, somTotalRegIds, somSetExpectedIds, somUniqueKey, somBeginPersistentIds, somEndPersistentIds

somTotalRegIds Function

Purpose

Returns the total number of SOM IDs that have been registered.

Syntax

unsigned long somTotalRegIds ( );

Description

The **somTotalRegIds** function returns the total number of SOM IDs that have been registered so far. This value can be used as a parameter to the **somSetExpectedIds** function to advise SOM about expected ID usage in later executions of a client program.

Parameters

None.

Return Value

Returns the total number of SOM IDs that have been registered.

C Example

```
#include <som.h>
main()
{ int i;
  somId id;
  somEnvironmentNew();
  id = somIdFromString(”abc”)
  i = somTotalRegIds();
  id = somIdFromString(”abc”);
  SOM_Test(i == somTotalRegIds);
}
```

Related Information

Functions: somCheckId, somRegisterId, somIdFromString, somStringFromId, somCompareIds, somSetExpectedIds, somUniqueKey, somBeginPersistentIds, somEndPersistentIds

somUniqueKey Function

Purpose

Returns the unique key associated with a SOM ID.

Syntax

unsigned long somUniqueKey (somID *id);*

Description

The **somUniqueKey** function returns the unique key associated with a SOM ID. The unique key for a SOM ID is a number that uniquely represents the string that the SOM ID represents. The unique key for a SOM ID is the same as the unique key for another SOM ID only if the two SOM IDs represent the same string.

Parameters

id The SOM ID for which the unique key is needed.

Return Value

An **unsigned long** representing the unique key of the specified SOM ID.

C Example

```
#include <som.h>
main()
{
   unsigned long k1, k2;
   k1 = somUniqueKey(somIdFromString(”abc”));
   k2 = somUniqueKey(somIdFromString(”abc”));
   SOM_Test(k1 == k2);
}
```

Related Information

Data Structures: somId (sombtype.h)

Functions: somCheckId, somRegisterId, somIdFromString, somStringFromId, somCompareIds, somTotalRegIds, somSetExpectedIds, somBeginPersistentIds, somEndPersistentIds

somVprintf Function

Purpose

Prints a formatted string in the manner of the C vprintf function.

Syntax

long somVprintf (string *fmt,* **va_list** *ap***);**

Description

The **somVprintf** function prints a formatted string using **SOMOutCharRoutine**, in the same manner as the C vprintf function. The implementation of **SOMOutCharRoutine** determines the destination of the output, while the C printf function is always directed to stdout. (The default output destination for **SOMOutCharRoutine** is stdout also, but this can be modified by the user.)

Parameters

fmt The format string to be output.

ap A **va_list** representing the values to be substituted into the format string.

Return Value

Returns the number of characters written.

C Example

```
#include <som.h>
main()
{
   va_list args = (va_list) SOMCalloc(20);
   va_list push = args;
   float f = 3.1415;
   char c = 'a';

   va_arg(push, int) = 1;
   va_arg(push, double) = f; /* note ANSI widening */
   va_arg(push, int) = c; /* here, too */
   va_arg(push, char*) = "this is a test";

   somVprintf("%d, %f, %c, %s\n", args);
}
```

Related Information

Data Structures: string (somcorba.h), **va_list** (stdarg.h)

Functions: somPrintf, somPrefixLevel, somLPrintf, SOMOutCharRoutine

SOMCalloc Function

Purpose

Allocates sufficient zeroed memory for an array of objects of a specified size.

Syntax

somToken (*SOMCalloc) (size_t *num,* **size_t** *size***);**

Description

The **SOMCalloc** function allocates an amount of memory equal to *num*size* (sufficient memory for an array of *num* objects of size *size).* The **SOMCalloc** function has the same interface as the C **calloc** function. It performs the same basic function as **calloc** with some supplemental error checking. If an error occurs, the **SOMError** function is called. This routine is replaceable by changing the value of the global variable **SOMCalloc.**

Parameters

num	The number of objects for which space is to be allocated.
size	The size of the objects for which space to is to be allocated.

Return Value

A pointer to the first byte of the allocated space.

Example

See function **somVprintf**.

Related Information

Data Structures: somToken (sombtype.h)

Functions: SOMMalloc, SOMRealloc, SOMFree

SOMClassInitFuncName Function

Purpose

Returns the name of the function used to initialize classes in a DLL.

Syntax

string (*SOMClassInitFuncName) ( );

Description

The **SOMClassInitFuncName** function is called by the SOM Class Manager to determine what function to call to initialize the classes in a DLL. The default version returns the string "SOMInitModule." The function can be replaced (so that the Class Manager will invoke a different function to initialize classes in a DLL) by changing the value of the global variable **SOMClassInitFuncName**.

Parameters

None.

Return Value

Returns the name of the function that should be used to initialize classes in a DLL.

C Example

```
#include <som.h>
string XYZFuncName() { return "XYZ"; }
main()
{
   SOMClassInitFuncName = XYZFuncName;
   ...
}
```

Related Information

Data Structures: string (somcorba.h)

Functions: SOMLoadModule, SOMDeleteModule

SOMDeleteModule Function

Purpose

Unloads a dynamically linked library (DLL).

Syntax

int (*SOMDeleteModule) (somToken *modHandle***);**

Description

The **SOMDeleteModule** function unloads the specified dynamically linked library (DLL). This routine is called by the SOM Class Manager to unload DLLs. **SOMDeleteModule** can be replaced (thus changing the way the Class Manager unloads DLLS) by changing the value of the global variable **SOMDeleteModule.**

Parameters

modHandle The **somToken** for the DLL to be unloaded. This token is supplied by the **SOMLoadModule** function when it loads the DLL.

Return Value

Returns 0 if successful or a non-zero system-specific error code otherwise.

Related Information

Data Structures: somToken (sombtype.h)

Functions: SOMLoadModule, SOMClassInitFuncName

SOMError Function

Purpose

Handles an error condition.

Syntax

void (*SOMError) (int *errorCode*, **string** *fileName*, **int** *lineNum***);**

Description

The **SOMError** function inspects the specified error code and takes appropriate action, depending on the severity of the error. The last digit of the error code indicates whether the error is classified as SOM_Fatal (9), SOM_Warn (2), or SOM_Ignore (1). The default implementation of **SOMError** prints a message that includes the specified error code, filename, and line number, and terminates the current process if the error is classified as SOM_Fatal. The *fileName* and *lineNum* arguments specify where the error occurred. This routine can be replaced by changing the value of the global variable **SOMError.**

For C and C++ programmers, SOM defines a convenience macro, **SOM_Error**, which invokes the **SOMError** function and supplies the last two arguments.

Parameters

errorCode	An integer representing the error code of the error.
fileName	The name of the file in which the error occurred.
lineNum	The line number where the error occurred.

Return Value

None.

Related Information

Macros: SOM_Test, SOM_TestC, SOM_WarnMsg, SOM_Assert, SOM_Expect, SOM_Error

SOMFree Function

Purpose

Frees the specified block of memory.

Syntax

void (*SOMFree) (somToken *ptr*);

Description

The **SOMFree** function frees the block of memory pointed to by *ptr.* SOMFree should only be called with a pointer previously allocated by **SOMMalloc** or **SOMCalloc.** The **SOMFree** function has the same interface as the C **free** function. It performs the same basic function as **free** with some supplemental error checking. If an error occurs, the **SOMError** function is called. This routine is replaceable by changing the value of the global variable **SOMFree.**

To free an *object* (rather than a block of memory), use the **somFree** method, rather than this function.

Parameters

ptr	A pointer to the block of storage to be freed.

Return Value

None.

C Example

```
#include <som.h>
main()
{
   somToken ptr = SOMMalloc(20);
   . . .
   somFree(ptr);
}
```

Related Information

Functions: SOMCalloc, SOMRealloc, SOMMalloc

Methods: somFree

SOMInitModule Function

Purpose

Invokes the class creation routines for the classes contained in an OS/2 or Windows class library (DLL).

Syntax

SOMEXTERN void SOMLINK SOMInitModule (
long *MajorVersion*,
long *MinorVersion*,
string *ClassName***);**

Description

On OS/2 or Windows, a class library (DLL) can contain the implementations for multiple classes, all of which should be created when the DLL is loaded. On OS/2, when loading a DLL, the SOM class manager determines the name of a DLL initialization function, and if the DLL exports a function of this name, the class manager invokes that function (whose purpose is to create the classes in the DLL). **SOMInitModule** is the default name for this DLL initialization function.

On Windows, the SOM class manager does *not* call **SOMInitModule**. It must be called from the default Windows DLL initialization function, LibMain. This call is made indirectly through the **SOM_ClassLibrary** macro (see the Example below).

Parameters

MajorVersion	The major version number of the class that was requested when the library was loaded.
MinorVersion	The minor version number of the class that was requested when the library was loaded.
ClassName	The name of the class that was requested when the library was loaded.

Return Value

None.

Example

```
#include "xyz.h"
#ifdef __IBMC__
  #pragma linkage (SOMInitModule, system)
#endif

SOMEXTERN void  SOMLINK SOMInitModule (long majorVersion,
                          long minorVersion, string className)
{
    SOM_IgnoreWarning (majorVersion);  /* This function makes  */
    SOM_IgnoreWarning (minorVersion);  /* no use of the passed */
    SOM_IgnoreWarning (className);     /* arguments.   */
    xyzNewClass (A_MajorVersion, A_MinorVersion);
}
```

For Windows, also include the following function:

```
#include <windows.h>
int CALLBACK LibMain (HINSTANCE inst,
                      WORD ds,
                      WORD Heapsize,
                      LPSTR cmdLine)
{
    SOM_IgnoreWarning (inst);
    SOM_ignoreWarning (ds);
    SOM_IgnoreWarning (heapSize);
    SOM_IgnoreWarning (cmdLine);

    SOM_ClassLibrary (”xyz.dll”);
    return 1;  /* Indicate success to loader */
}
```

Related Information

Functions: SOMClassInitFuncName

Methods: somGetInitFunction

Macros: SOM_ClassLibrary

SOMLoadModule Function

Purpose

Loads the dynamically linked library (DLL) containing a SOM class.

Syntax

int (*SOMLoadModule) (
string *className*,
string *fileName,*
string *functionName,*
long *majorVersion,*
long *minorVersion,*
somToken ****modHandle*);**

Description

The **SOMLoadModule** function loads the dynamically linked library (DLL) containing a SOM class. This routine is called by the SOM Class Manager to load DLLs. **SOMLoadModule** can be replaced (thus changing the way the Class Manager loads DLLs) by changing the value of the global variable **SOMLoadModule.**

Parameters

className	The name of the class whose DLL is to be loaded.
fileName	The name of the DLL library file. This can be either a simple name or a fully qualified pathname.
functionName	The name of the routine to be called after the DLL is loaded. The routine is responsible for creating a class object for each class in the DLL. Typically, this argument will have the value **SOMInitModule,** obtained from the **SOMClassInitFuncName** function. If no **SOMInitModule** entry exists in the DLL, the default version of **SOMLoadModule** looks for a routine named **<***className***>NewClass** instead. If neither entry point is found, the default version of **SOMLoadModule** fails.
majorVersion	The expected major version number of the class, to be passed to the initialization routine of the DLL.
minorVersion	The expected minor version number of the class, to be passed to the initialization routine of the DLL.
modHandle	The address where **SOMLoadModule** should place a token that can be subsequently used by the **SOMDeleteModule** routine to unload the DLL.

Return Value

Returns 0 if successful or a non-zero system-specific error code otherwise.

Related Information

Functions: SOMDeleteModule, SOMClassInitFuncName

SOMMalloc Function

Purpose

Allocates the specified amount of memory.

Syntax

somToken (*SOMMalloc) (size_t *size***);**

Description

The **SOMMalloc** function allocates *size* bytes of memory. The **SOMMalloc** function has the same interface as the C **malloc** function. It performs the same basic function as **malloc** with some supplemental error checking. If an error occurs, the **SOMError** function is called. This routine is replaceable by changing the value of the global variable **SOMMalloc.**

Parameters

size	The amount of memory to be allocated, in bytes.

Return Value

A pointer to the first byte of the allocated space.

Example

See function **SOMFree**.

Related Information

Functions: SOMCalloc, SOMRealloc, SOMFree

SOMOutCharRoutine Function

Purpose

Prints a character. This function is replaceable.

Syntax

int (*SOMOutCharRoutine) (char *c***);**

Description

SOMOutCharRoutine is a replaceable character output routine. It is invoked by SOM whenever a character is generated by one of the SOM error-handling or debugging macros. The default implementation outputs the specified character to stdout. To change the destination of character output, store the address of a user-written character output routine in global variable **SOMOutCharRoutine**.

Another function, **somSetOutChar**, may be preferred over the **SOMOutCharRoutine** function. The **somSetOutChar** function enables each application (or thread) to have a customized character output routine.

Parameters

c	The character to be output.

Return Value

Returns 0 if an error occurs and 1 otherwise.

Example

```
#include <som.h>
#pragma linkage(myCharacterOutputRoutine, system)
/* Define a replacement routine: */
int SOMLINK myCharacterOutputRoutine (char c)
{
    (Customized code goes here)
}
...
/* After the next stmt all output */
/* will be sent to the new routine   */
SOMOutCharRoutine = myCharacterOutputRoutine;
```

Related Information

Functions: somVprintf, somPrefixLevel, somLPrintf, somPrintf, somSetOutChar

SOMRealloc Function

Purpose

Changes the size of a previously allocated region of memory.

Syntax

somToken (*SOMRealloc) (somToken *ptr,* **size_t** *size***);**

Description

The **SOMRealloc** function changes the size of the previously allocated region of memory pointed to by *ptr* so that it contains *size* bytes. The new size may be greater or less than the original size. The **SOMRealloc** function has the same interface as the C **realloc** function. It performs the same basic function as **realloc** with some supplemental error checking. If an error occurs, the **SOMError** function is called. This routine is replaceable by changing the value of the global variable **SOMRealloc.**

Parameters

ptr — A pointer to the previously allocated region of memory. If NULL, a new region of memory of *size* bytes is allocated.

size — The size in bytes for the re-allocated storage. If zero, the memory pointed to by *ptr* is freed.

Return Value

A pointer to the first byte of the re-allocated space. (A pointer is returned because the block of storage may need to be moved to increase its size.)

Related Information

Functions: SOMCalloc, SOMMalloc, SOMFree

SOM_Assert Macro

Purpose

Asserts that a **boolean** condition is true.

Syntax

void SOM_Assert (
boolean *condition,*
long *errorCode***);**

Description

The **SOM_Assert** macro is used to place **boolean** assertions in a program:

- If *condition* is FALSE, and *errorCode* indicates a warning-level error and **SOM_WarnLevel** is set to be greater than zero, then a warning message is output.
- If *condition* is FALSE and *errorCode* indicates a fatal error, an error message is output and the process is terminated.
- If *condition* is TRUE and **SOM_AssertLevel** is set to be greater than zero, then an informational message is output.

External (Global) Data

```
long SOM_WarnLevel;  /* default = 0 */
long SOM_AssertLevel; /* default 0 */
```

Parameters

condition	A **boolean** expression that is expected to be TRUE (nonzero).
errorCode	The integer error code for the error to be raised if *condition* is FALSE.

Expansion

If *condition* is FALSE, and *errorCode* indicates a warning-level error and **SOM_WarnLevel** is set to be greater than zero, then a warning message is output. If *condition* is FALSE and *errorCode* indicates a fatal error, an error message is output and the process is terminated. If *condition* is TRUE and **SOM_AssertLevel** is set to be greater than zero, then an information message is output.

Example

```
#include <som.h>
main()
{
   SOM_WarnLevel = 1;
   SOM_Assert(2==2, 29);
}
```

Related Information

Macros: SOM_Expect, SOM_Test, SOM_TestC

SOM_ClassLibrary Macro

Purpose

Identifies the file name of the DLL for a SOM class library in a Windows LibMain function.

Syntax

void SOM_ClassLibrary (string "*libname.dll***");**

Description

Each Windows SOM class library must supply a Windows LibMain function. In LibMain, the **SOM_ClassLibrary** macro identifies both the actual file name of the library as it would appear in a Windows LoadLibrary call and the location of the library's **SOMInitModule** function. This information is passed to the SOM Kernel, which in turn registers the library and schedules the execution of the **SOMInitModule** function. This macro can also be used in OS/2 class libraries within the context of a DLL "init/term" function

Typically, the SOM Kernel invokes the **SOMInitModule** function of each statically loaded class library during the execution of the **somMainProgram** function in the using application. For dynamically loaded class libraries, **SOMInitModule** is invoked immediately upon completion of the library's LibMain (or an OS/2 DLL "init/term") function.

Because the **SOM_ClassLibrary** macro expands to reference the **SOMInitModule** function, either a declaration of the **SOMInitModule** function, or the function itself, should precede the appearance of **SOM_ClassLibrary** in the current compilation unit, as shown in the example below).

Parameters

libname.dll	The name of the file containing the DLL (as the name would appear in a Windows LoadLibrary call).

Example

```
/* This example illustrates the use of the SOM_ClassLibrary
   macro in a Windows LibMain function */

#include <som.h>
SOMEXTERN void SOMLINK SOMInitModule (long majorVersion,
                                      long minorVersion,
                                      string className);

#include <windows.h>
int CALLBACK LibMain (HINSTANCE inst,
                      WORD ds,
                      WORD Heapsize,
                      LPSTR cmdLine)
{
    SOM_IgnoreWarning (inst);
    SOM_ignoreWarning (ds);
    SOM_IgnoreWarning (heapSize);
    SOM_IgnoreWarning (cmdLine);

    SOM_ClassLibrary ("xyz.dll");
    return 1;  /* Indicate success to loader */
}
```

Related Information

Macros: SOM_MainProgram

Functions: somMainProgram

SOM_CreateLocalEnvironment Macro

Purpose

Creates and initializes a local **Environment** structure.

Syntax

Environment * SOM_CreateLocalEnvironment ( );

Description

The **SOM_CreateLocalEnvironment** macro creates a local **Environment** structure. This **Environment** structure can be passed to methods as the **Environment** argument so that exception information can be returned without affecting the global environment.

Parameters

None.

Expansion

The **SOM_CreateLocalEnvironment** expands to an expression of type (**Environment ***).

C Example

```
Environment *ev;
ev = SOM_CreateLocalEnvironment();
_myMethod(obj, ev);
...
SOM_DestroyLocalEnvironment(ev);
```

Related Information

Data Structures: Environment (somcorba.h)

Macros: SOM_DestroyLocalEnvironment, SOM_InitEnvironment, SOM_UninitEnvironment

Functions: somGetGlobalEnvironment

SOM_DestroyLocalEnvironment Macro

Purpose

Destroys a local **Environment** structure.

Syntax

void SOM_DestroyLocalEnvironment (Environment * *ev***);**

Description

The **SOM_DestroyLocalEnvironment** macro destroys a local **Environment** structure, such as one created using the **SOM_CreateLocalEnvironment** macro.

Parameters

ev	A pointer to the **Environment** structure to be discarded.

Expansion

The **SOM_DestroyLocalEnvironment** function first invokes the **somExceptionFree** function on the **Environment** structure; then it invokes **SOMFree** on it to free the memory it occupies.

Example

```
Environment *ev;
ev = SOM_CreateLocalEnvironment();
_myMethod(obj, ev);
....
SOM_DestroyLocalEnvironment(ev);
```

Related Information

Macros: SOM_CreateLocalEnvironment, SOM_UninitEnvironment

Functions: somExceptionFree

SOM_Error Macro

Purpose

Reports an error condition.

Syntax

void SOM_Error (long *errorCode***);**

Description

The **SOM_Error** macro invokes the **SOMError** error handling procedure with the specified error code, supplying the filename and line number where the macro was invoked. The default implementation of **SOMError** outputs a message containing the error code, filename, and line number. Additionally, if the last digit of the error code indicates a serious error (that is, value SOM_Fatal), the process is terminated.

Parameters

errorCode	The integer error code for the error to be reported.

Expansion

The **SOM_Error** macro invokes the **SOMError** error handler, supplying the filename and line number where the macro was invoked.

Related Information

Functions: SOMError

SOM_Expect Macro

Purpose

Asserts that a **boolean** condition is expected to be true.

Syntax

void SOM_Expect (boolean *condition***);**

Description

The **SOM_Expect** macro is used to place **boolean** assertions that are expected to be true into a program:

- If *condition* is FALSE and **SOM_WarnLevel** is set to be greater than zero, then a warning message is output.
- If *condition* is TRUE and **SOM_AssertLevel** is set to be greater than zero, then an informational message is output.

Parameters

condition	A boolean expression that is expected to be TRUE (nonzero).

Expansion

If *condition* is FALSE and **SOM_WarnLevel** is set to be greater than zero, then a warning message is output. If *condition* is TRUE and **SOM_AssertLevel** is set to be greater than zero, then an information message is output.

Example

```
SOM_Expect(2==2);
```

Related Information

Macros: SOM_Assert, SOM_Test, SOM_TestC

SOM_GetClass Macro

Purpose

Returns a pointer to the class object of which a SOM object is an instance.

Syntax

SOMClass SOM_GetClass (SOMObject *objPtr***);**

Description

The **SOM_GetClass** macro returns the class object of which *obj* is an instance. This is done without recourse to a method call on the object. The **somGetClass** method introduced by **SOMObject** is also intended to return the class of which an object is an instance, and the default implementation provided for this method by **SOMObject** uses the macro.

Important Note: It is generally recommended that the **somGetClass** method call be used, since it cannot be known whether the class of an object wishes to provide special handling when its address is requested from an instance. But, there are (rare) situations where a method call cannot be made, and this macro can then be used. If you are unsure as to whether to use the method or the macro, you should use the method.

Parameters

objPtr	A pointer to the object whose class is needed.

C++ Example

```
#include <somcls.xh>
#include <animal.xh>
main()
{
   Animal *a = new Animal;
   SOMClass cls1 = SOM_GetClass(a);
   SOMClass cls2 = a->somGetClass();
   if (cls1 == cls2)
      printf(”macro and method for getClass the same for Animal\n”);
   else
      printf(”macro and method for getClass not same for Animal\n”);
}
```

Related Information

Methods: somGetClass

SOM_InitEnvironment Macro

Purpose

Initializes a local **Environment** structure.

Syntax

void SOM_InitEnvironment (Environment * *ev***);**

Description

The **SOM_InitEnvironment** macro initializes a locally declared **Environment** structure. This **Environment** structure can then be passed to methods as the **Environment** argument so that exception information can be returned without affecting the global environment.

Parameters

ev	A pointer to the **Environment** structure to be initialized.

Expansion

The **SOM_InitEnvironment** initializes an **Environment** structure to zero.

C Example

```
Environment ev;
SOM_InitEnvironment(&ev);
_myMethod(obj, &ev);
....
SOM_UninitEnvironment(&ev);
```

Related Information

Macros: SOM_DestroyLocalEnvironment, SOM_CreateLocalEnvironment, SOM_UninitEnvironment

Functions: somGetGlobalEnvironment

SOM_MainProgram Macro

Purpose

Identifies an application as a SOM program and registers an end-of-program exit procedure to release SOM resources when the application terminates.

Syntax

SOMClassMgr SOM_MainProgram ( );

Description

The **SOM_MainProgram** macro should appear near the beginning of each Windows application program that uses SOM or a SOM class library. It can also be used in OS/2 or AIX programs but is not generally required on these platforms. Any statically referenced SOM class libraries are initialized during the execution of this macro, and an end-of-program exit procedure is established to release SOM resources during normal program termination. (This macro combines the execution of the C/C++ "atexit" function with the SOM **somMainProgram** function and returns a reference to the global **SOMClassMgr** object.)

Parameters

None.

Example

```
#include <som.h>
#include <windows.h>

int PASCAL WinMain (HINSTANCE inst,
                    WORD ds,
                    WORD Heapsize,
                    LPSTR cmdLine)
{
    ...
    SOM_MainProgram ();
    ...
    /* Rest of main program follows */
}
```

Related Information

Functions: somMainProgram

Macros: SOM_ClassLibrary

SOM_NoTrace Macro

Purpose

Used to turn off method debugging.

Syntax

SOM_NoTrace (<token> *className,* **<token>** *methodName***);**

Description

The **SOM_NoTrace** macro is used to turn off method debugging. Within an implementation file for a class, before #including the implementation (.ih or .xih) header file for the class, #define the <*className*>**MethodDebug** macro to be **SOM_NoTrace**. Then, <*className*>**MethodDebug** will have no effect.

Parameters

className The name of the class for which tracing will be turned off, given as a simple token rather than a quoted string.

methodName The name of the method for which tracing will be turned off, given as a simple token rather than a quoted string.

Expansion

The **SOM_NoTrace** macro has a null (empty) expansion.

Example

Within an implementation file:

```
#define AnimalMethodDebug(c,m) SOM_NoTrace(c,m)
#include <animal.ih>
/* Now AnimalMethodDebug does nothing */
```

SOM_ParentNumResolve Macro

Purpose

Obtains a pointer to a method procedure from a list of method tables. Used by C and C++ implementation bindings to implement parent method calls.

Syntax

somMethodPtr SOM_ParentNumResolve (
<token> *IntroClass,*
long *parentNum,*
somMethodTabs *parentMtabs,*
<token> *methodName***);**

Description

The **SOM_ParentNumResolve** macro invokes the **somParentNumResolve** function to obtain a pointer to the static method procedure that implements the specified method for the specified parent. The method is specified by indicating the introducing class, *IntroClass*, and the method name, *methodName*.

Parameters

introClass The name of the class that introduces *methodName*. This name should be given as a simple token, rather than a quoted string (for example, *Animal* rather than "*Animal*").

parentNum The position of the desired parent. The first (leftmost) parent of a class has position 1.

parentMtabs A list of parent method tables acquired by invoking the **somGetPClsMtabs** method on a class object.

methodName The name of the method to be resolved. This name should be given as a simple token, rather than a quoted string (for example, *setSound* rather than "*setSound*").

Expansion

The expansion of the macro produces an expression that is appropriately typed for application of the evaluated result to the indicated method's arguments, as illustrated below.

Example

```
#include <somcls.h>

main()
{
 SOMClassMgr *cm = somEnvironmentNew();
 somMethodTabs mList = _somGetPClsMtabs(_SOMClass);
 SOM_ParentNumResolve(SOMObject, 1, mList, somDumpSelfInt)
      (_SOMClass,1);
}
```

Related Information

Functions: somParentNumResolve

Methods: somGetPClsMtabs

SOM_Resolve Macro

Purpose

Obtains a pointer to a static method procedure.

Syntax

somMethodPtr SOM_Resolve (
SOMObject *objPtr,*
<token> *className,*
<token> *methodName***);**

Description

The **SOM_Resolve** macro invokes the **somResolve** function to obtain a pointer to the static method procedure that implements the specified method for the specified object. This pointer can be used for efficient repeated casted method invocations on instances of the class of the object on which the resolution is done, or instances of subclasses of this class. The name of the class that introduces the method and the name of the method must be known to use this macro. Otherwise, use the **somResolveByName, somFindMethod** or **somFindMethodOk** method.

The **SOM_Resolve** macro can only be used to obtain a method procedure for a static method (one defined in the IDL specification for a class); not a dynamic method. Unlike the **SOM_ResolveNoCheck** macro, the **SOM_Resolve** macro performs several consistency checks on the object pointed to by *objPtr.*

Parameters

objPtr	A pointer to the object to which the resolved method procedure will be applied.
className	The name of the class that introduces *methodName*. This name should be given as a simple token, rather than a quoted string (for example, *Animal* rather than "*Animal*").
methodName	The name of the method to be resolved. This name should be given as a simple token, rather than a quoted string (for example, *setSound* rather than "*setSound*").

Expansion

The **SOM_Resolve** macro uses the *className* and *methodName* to construct the method token for the specified method, then invokes the **somResolve** function. Thus, the macro expands to an expression that represents the entry-point address of the method procedure. This value can be stored in a variable and used for subsequent invocations of the method.

Example

```
Animal myObj = AnimalNew();
somMethodProc *procPtr;
procPtr = SOM_Resolve(myObj, Animal, setSound);
/* note that procPtr will need to be typecast when it is used */
```

Related Information

Macros: SOM_ResolveNoCheck

Functions: somResolve, somClassResolve, somResolveByName

Methods: somFindMethod, somFindMethodOk, somDispatch, somClassDispatch

SOM_ResolveNoCheck Macro

Purpose

Obtains a pointer to a static method procedure, without doing consistency checks.

Syntax

somMethodPtr SOM_ResolveNoCheck (
SOMObject *objPtr,*
<token> *className,*
<token> *methodName***);**

Description

The **SOM_ResolveNoCheck** macro invokes the **somResolve** function to obtain a pointer to the method procedure that implements the specified method for the specified object. This pointer can be used for efficient repeated invocations of the same method on the same type of objects. The name of the class that introduces the method and the name of the method must be known at compile time. Otherwise, use the **somFindMethod** or **somFindMethodOk** method.

The **SOM_ResolveNoCheck** macro can only be used to obtain a method procedure for a static method (one defined in the IDL specification for a class) and not a method added to a class at run time. Unlike the **SOM_Resolve** macro, the **SOM_ResolveNoCheck** macro does not perform any consistency checks on the object pointed to by *objPtr.*

Parameters

objPtr	A pointer to the object to which the resolved method procedure will be applied.
className	The name of the class that introduces *methodName*. This name should be given as a simple token, rather than a quoted string (for example, *Animal* rather than "*Animal*").
methodName	The name of the method to be resolved. This name should be given as a simple token, rather than a quoted string (for example, *setSound* rather than "*setSound*").

Expansion

The **SOM_ResolveNoCheck** macro uses the *className* and *methodName* to construct an expression whose value is the method token for the specified method, then invokes the **somResolve** function. Thus, the macro expands to an expression that represents the entry-point address of the method procedure. This value can be stored in a variable and used for subsequent invocations of the method.

Example

```
Animal myObj = AnimalNew();
somMethodProc *procPtr;
procPtr = SOM_ResolveNoCheck(myObj, Animal, setSound)
```

Related Information

Macros: SOM_Resolve

Functions: somResolve, somClassResolve, somResolveByName

Methods: somDispatch, somClassDispatch, somFindMethod, somFindMethodOk

SOM_SubstituteClass Macro

Purpose

Provides a convenience macro for invoking the **somSubstituteClass** method.

Syntax

long SOM_SubstituteClass (
<token> *oldClass***,**
<token> *newClass***);**

Description

The method **somSubstituteClass** requires existing class objects as arguments. Therefore, the macro **SOM_SubstituteClass** first assures that the classes named *oldClass* and *newClass* exist, and then calls the method **somSubstituteClass** with these class objects as arguments.

Parameters

oldClass — The name of the class to be substituted, given as a simple token rather than a quoted string.

newClass — The name of the class that will replace *oldClass*, given as a simple token rather than a quoted string.

Example

See the method **somSubstituteClass**.

Related Information

Methods: somSubstituteClass

SOM_Test Macro

Purpose

Tests whether a **boolean** condition is true; if not, a fatal error is raised.

Syntax

void SOM_Test (boolean *expression***);**

Description

The **SOM_Test** macro tests the specified **boolean** expression:

- If the expression is TRUE and **SOM_AssertLevel** is set to a value greater than zero, then an information message is output.
- If the expression is FALSE, an error message is output and the process is terminated. Note: The **SOM_TestC** macro is similar, except that it only outputs a warning message in this situation.

Parameters

expression The **boolean** expression to test.

External (Global) Data

```
long SOM_AssertLevel;  /* default is 0 */
```

Expansion

The **SOM_Test** macro tests the specified boolean expression. If the expression is TRUE and **SOM_AssertLevel** is set to a value greater than zero, then an information message is output. If the expression is FALSE, an error message is output and the process is terminated.

C Example

```
#include <som.h>
main()
{
   SOM_AssertLevel = 1;
   SOM_Test(1=1);
}
```

Related Information

Macros: SOM_Expect, SOM_Assert, SOM_TestC

SOM_TestC Macro

Purpose

Tests whether a **boolean** condition is true; if not, a warning message is output.

Syntax

void SOM_TestC (boolean *expression***);**

Description

The **SOM_TestC** macro tests the specified **boolean** expression:

- If the expression is TRUE and **SOM_AssertLevel** is set to a value greater than zero, then an information message is output.
- If the expression is FALSE and **SOM_WarnLevel** is set to a value greater than zero, then a warning message is output. Note: The **SOM_Test** macro is similar, except that it raises a fatal error in this situation.

Parameters

expression The **boolean** expression to test.

External (Global) Data

```
long SOM_AssertLevel;  /* default is 0 */

long SOM_WarnLevel; /* default is 0 */
```

Expansion

The **SOM_TestC** macro tests the specified **boolean** expression. If the expression is TRUE and **SOM_AssertLevel** is set to a value greater than zero, then an information message is output. If the expression is FALSE and **SOM_WarnLevel** is set to a value greater than zero, a warning message is output.

C Example

```
#include <som.h>
main()
{
   SOM_WarnLevel = 1;
   SOM_TestC(1=1);
}
```

Related Information

Macros: SOM_Expect, SOM_Assert, SOM_Test

SOM_UninitEnvironment Macro

Purpose

Uninitializes a local **Environment** structure.

Syntax

void SOM_UninitEnvironment (Environment * *ev***);**

Description

The **SOM_UninitEnvironment** macro uninitializes a locally declared **Environment** structure.

Parameters

ev	A pointer to the **Environment** structure to be uninitialized.

Expansion

The **SOM_UninitEnvironment** invokes the **somExceptionFree** function on the specified **Environment** structure.

C Example

```
Environment ev;
SOM_InitEnvironment(&ev);
_myMethod(obj, &ev);
...
SOM_UninitEnvironment(&ev);
```

Related Information

Macros: SOM_DestroyLocalEnvironment, SOM_InitEnvironment

SOM_WarnMsg Macro

Purpose

Reports a warning message.

Syntax

void SOM_WarnMsg (string *msg***);**

Description

If **SOM_WarnLevel** is set to a value greater than zero, the **SOM_WarnMsg** macro prints the specified message, along with the filename and line number where the macro was invoked.

Parameters

msg	The warning message to be output.

Expansion

If **SOM_WarnLevel** is set to a value greater than zero, the **SOM_WarnMsg** macro prints the specified message, along with the filename and line number where the macro was invoked.

Related Information

Macros: SOM_Error

SOMClass Class

Description

SOMClass is the root class for all SOM metaclasses. That is, all SOM metaclasses must be subclasses of **SOMClass** or some other class derived from it. It defines the essential behavior common to all SOM classes. In particular, it provides a suite of methods for initializing class objects, generic methods for manufacturing instances of those classes, and methods that dynamically obtain or update information about a class and its methods at run time.

Just as all SOM classes are expected to have **SOMObject** (or a class derived from **SOMObject**) as their base class, all SOM classes are expected to have **SOMClass** or a class derived from **SOMClass** as their metaclass. Metaclasses define "class" methods (sometimes called "factory" methods or "constructors") that manufacture objects from any class object that is defined as an instance of the metaclass.

To define your own class methods, define your own metaclass by subclassing **SOMClass** or one of its subclasses. Three methods that **SOMClass** inherits and overrides from **SOMObject** are typically overridden by any metaclass that introduces instance data — **somInit**, **somUninit**, and **somDumpSelfInt**. The new methods introduced in **SOMClass** that are frequently overridden are **somNew**, **somRenew**, and **somClassReady**. (See the descriptions of these methods for further information.)

Other reasons for creating a new metaclass include tracking object instances, automatic garbage collection, interfacing to a persistent object store, or providing/managing information that is global to a set of object instances.

File Stem

somcls

Base

SOMObject

Metaclass

SOMClass (**SOMClass** is the only class with itself as metaclass.)

Ancestor Classes

SOMObject

Types

```
typedef sequence <SOMClass> SOMClassSequence;

struct  somOffsetInfo  {
     SOMClass          cls;
     long              offset
     };
typedef sequence <somOffsetInfo> SOMOffsets;
```

New Methods

Attributes:

readonly attribute somOffsets somInstanceDataOffsets

_get_somInstanceDataOffsets returns a sequence of structures, each of which indicates an ancestor of the receiver class (or the receiver class itself) and the offset to the beginning of the instance data introduced by the indicated class in an instance of the receiver class. The **somOffsets** information can be used in conjunction with information derived from calls to a *SOM Interface Repository* to completely determine the layout of SOM objects at runtime.

C++ Example

```
#include <somcls.xh>
main()
{
   int i;
   SOMClassMgr *scm = somEnvironmentNew();
   somOffsets so = _SOMClass->_get_somInstanceDataOffsets();
   for (i=0; i<so._length; i++)
      printf("In an instance of SOMClass, %s data starts at %d\n",
                so._buffer[i]->cls->somGetName(),
                so._buffer[i]->offset);
}
```

Introduced Methods

Group: Instance Creation (Factory)

somAllocate
somDeallocate
somNew
somNewNoInit
somRenew
somRenewNoInit
somRenewNoInitNoZero
somRenewNoZero

Group: Initialization/Termination

somAddDynamicMethod
somClassReady

Group: Access

somGetInstancePartSize
somGetInstanceSize
somGetInstanceToken
somGetMemberToken
somGetMethodData
somGetMethodDescriptor
somGetMethodIndex
somGetMethodToken
somGetName
somGetNthMethodData
somGetNthMethodInfo
somGetNumMethods
somGetNumStaticMethods
somGetParents
somGetVersionNumbers

Group: Testing

somCheckVersion
somDescendedFrom
somSupportsMethod

Group: Dynamic

somFindMethod
somFindMethodOk
somFindSMethod
somFindSMethodOk
somLookupMethod

Overridden Methods

somDumpSelfInt
somDefaultInit
somDestruct

Deprecated Methods

Use of the methods listed below is discouraged. There are three reasons for this:

First, these methods are used in constructing classes, and this capability is provided by the function **somBuildClass**. Class construction in SOM is currently a fairly complex activity, and it is likely to become even more so as the SOMobjects kernel evolves. To avoid breaking source code that constructs classes, you are advised to always use **somBuildClass** to build SOM classes. Note: The SOM language bindings always use **somBuildClass**.

Second, these methods are used for customizing aspects of SOM classes, such as method resolution and object creation. Doing this requires that metaclasses override various methods introduced by **SOMClass**. However, if this is done without the Cooperation Framework that implements the SOM Metaclass Framework, SOMobjects cannot guarantee that applications will function correctly. Unfortunately, the Cooperation Framework (while available to SOM users as an experimental feature) is not officially supported by the SOMobjects Toolkit. So, this is another reason why the following methods are deprecated.

Finally, some of these methods are now obsolete, so it seems appropriate that their use be discouraged.

somAddStaticMethod
somGetApplyStub
somGetClassData
somGetClassMtab
somGetInstanceOffset
somGetMethodOffset
somGetParent
somGetPClsMtab
somGetPClsMtabs
somGetRdStub
somInitClass
somInitMIClass
somOverrideMtab
somOverrideSMethod
somSetClassData
somSetMethodDescriptor
_get_somDirectInitClasses attribute
_set_somDirectInitClasses attribute

somAddDynamicMethod Method

Purpose

Adds a new dynamic instance method to a class. Dynamic methods are not part of the declared interface to a class of objects, and are therefore not supported by implementation and usage bindings. Instead, dynamic methods provide a way to dynamically add new methods to a class of objects during execution. SOM provides no standard protocol for informing a user of the existence of dynamic methods and the arguments they take. Dynamic methods must be invoked using name-lookup or dispatch resolution.

IDL Syntax

void somAddDynamicMethod (
in somId *methodId***,**
in somId *methodDescriptor***,**
in somMethodPtr *method***,**
in somMethodPtr *applyStub***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somAddDynamicMethod** method adds a new dynamic instance method to the receiving class. This involves recording the method's ID, descriptor, method procedure (specified by *method*), and apply stub in the receiving class's method data.

The arguments to **somAddDynamicMethod** should be non-null and obey the requirements expressed below. This is the responsibility of the implementor of a class, who in general has no knowledge of whether clients of this class will require use of the *applyStub* argument.

Parameters

receiver A pointer to a SOM class object.

methodId A **somId** that names the method.

methodDescriptor
A **somId** appropriate for requesting information concerning the method from the SOM IR. This is currently of the form <className>::<methodName>.

method A pointer to the procedure that will implement the new method. The first argument of this procedure is the address of the object on which it is being invoked.

applyStub A pointer to a procedure that returns nothing and receives as arguments: a method receiver; an address where the return value from the method call is to be stored; a pointer to a method procedure; and a va_list containing the arguments to the method. The applyStub procedure (which is usually called by **somDispatch**) must unload its va_list argument into separate variables of the correct type for the method, invoke its procedure argument on these variables, and then copy the result of the procedure invocation to the address specified by the return value argument.

Return Value

None.

C Example

```
/* New dynamic method ”newMethod1” for class ”XXX” */
static char *somMN_newMethod1 = ”newMethod1”;
static somId somId_newMethod1 = &somMN_newMethod1;
static char *somDS_newMethod1 = ”XXX::newMethod1”;
static somId somDI_newMethod1 = &somDS_newMethod1;

static void SOMLINK somAP_newMethod1(SOMObject somSelf,
                                     void *__retVal,
                                     somMethodProc *__methodPtr,
                                     va_list __ap)
{
   void* __somSelf = va_arg(__ap, SOMObject);
   int arg1 = va_arg(__ap, int);
   SOM_IgnoreWarning(__retVal);
   ((somTD_SOMObject_newMethod1) __methodPtr) (__somSelf, arg1);
}

main()
{
   _somAddDynamicMethod (
   XXXClassData.classObject,           /* Receiver (class object) */
   somId_newMethod1,                   /* method name somId       */
   somDI_newMethod1,                   /* method descriptor somId */
   (somMethodProc *) newMethod1,       /* method procedure        */
   (somMethodProc *) somAP_newMethod1); /* method apply stub      */
}
```

Original Class

SOMClass

Related Information

Methods: somGetMethodDescriptor

somAllocate Method

Purpose

Supports class-specific memory allocation for class instances. Cannot be overridden.

IDL Syntax

string somAllocate (in long *size***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

When building a class, the **somBuildClass** function is responsible for registering the procedure that will be executed when this method is invoked on the class. The default procedure registered by **somBuildClass** uses the **SOMMalloc** function, but the IDL modifier **somallocate** can be used in the SOM IDL class implementation section to indicate a different procedure. Users of this method should be sure to use the dual method, **somDeallocate**, to free allocated storage. Also, if the IDL modifier **somallocate** is used to indicate a special allocation routine, the IDL modifier **somdeallocate** should be used to indicate a dual procedure to be called when the **somDeallocate** method is invoked.

Parameters

receiver	A pointer to the class object whose memory allocation method is desired.
size	The number of bytes to be allocated.

Return Value

string	A pointer to the first byte of the allocated memory region, or NULL if sufficient memory is not available.

C++ Example

```
#include <som.xh>
#include <somcls.xh>
main()
{
   SOMClassMgr *cm = somEnvironmentNew();
   /* Use SOMClass's instance allocation method */
   string newRegion = _SOMClass->somAllocate(20);
}
```

Original Class

SOMClass

Related Information

Methods: somDeallocate

somCheckVersion Method

Purpose

Checks a class for compatibility with the specified major and minor version numbers. Not generally overridden.

IDL Syntax

boolean somCheckVersion (
In long *majorVersion*,
In long *minorVersion*);

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somCheckVersion** method checks the receiving class for compatibility with the specified major and minor version numbers. An implementation is compatible with the specified version numbers if it has the same major version number and a minor version number that is equal to or greater than *minorVersion*. The version number pair (0,0) is considered to match any version.

This method is called automatically after creating a class object to verify that a dynamically loaded class definition is compatible with a client application.

Parameters

receiver	A pointer to the SOM class whose version information should be checked.
majorVersion	This value usually changes only when a significant enhancement or incompatible change is made to a class.
minorVersion	This value changes whenever minor enhancements or fixes are made to a class. Class implementors usually maintain downward compatibility across changes in the *minorVersion* number.

Return Value

Returns 1 (true) if the implementation of this class is compatible with the specified major and minor version number, and 0 (false) otherwise.

C Example

```
#include <animal.h>
main()
{
  Animal myAnimal;
  myAnimal = AnimalNew();

  if (_somCheckVersion(_Animal, 0, 0))
     somPrintf("Animal IS compatible with 0.0\n");
  else
     somPrintf("Animal IS NOT compatible with 0.0\n");

  if (_somCheckVersion(_Animal, 1, 1))
     somPrintf("Animal IS compatible with 1.1\n");
  else
     somPrintf("Animal IS NOT compatible with 1.1\n");

  _somFree(myAnimal);
}
```

Assuming that the implementation of Animal is version 1.0, this program produces the following output:

```
Animal IS compatible with 0.0
Animal IS NOT compatible with 1.1
```

Original Class

SOMClass

somClassReady Method

Purpose

Indicates that a class has been constructed and is ready for normal use. Designed to be overridden.

IDL Syntax

void somClassReady ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somClassReady** method is invoked automatically by the **somBuildClass** function after constructing and initializing a class object. The default implementation of this method provided by **SOMClass** simply registers the newly constructed class with **SOMClassMgrObject**. Metaclasses can override this method to augment class construction with additional registration protocol.

To have special processing done when a class object is created, you must define a metaclass for the class that overrides **somClassReady**. The final statement in any overriding method should invoke the parent method to ensure that the class is properly registered with **SOMClassMgrObject**. Users of the C and C++ implementation bindings for SOM classes should never invoke the **somClassReady** method directly; it is invoked automatically during class construction.

Parameters

receiver	A pointer to the class object that should be registered.

Return Value

None.

Original Class

SOMClass

somDeallocate Method

Purpose

Frees memory originally allocated by the **somAllocate** method from the same class object. Cannot be overridden.

IDL Syntax

void somDeallocate (in string *memPtr***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somDeallocate** method is intended for use to free memory allocated using its dual method, **somAllocate**. When building a class, the **somBuildClass** function is responsible for registering the procedure that will be executed when this method is invoked on the class. The default procedure registered by **somBuildClass** uses the **SOMFree** function, but the IDL modifier **somdeallocate** can be used in the SOM IDL class implementation section to indicate a different procedure. Users of this method should be sure that the dual method, **somAllocate**, was originally used to allocate storage. Also, if the IDL modifier **somdeallocate** is used to indicate a special deallocation routine, the IDL modifier **somallocate** should be used to indicate a dual procedure to be called when **somAllocate** is invoked.

Parameters

receiver	A pointer to the class object whose **somAllocate** was originally used to allocate the memory now to be freed.
memPtr	A pointer to the first byte of the region of memory that is to be freed.

Return Value

None.

Original Class

SOMClass

Related Information

Methods: somAllocate

somDescendedFrom Method

Purpose

Tests whether one class is derived from another. Not generally overridden.

IDL Syntax

boolean somDescendedFrom (in SOMClass *aClassObj***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

Tests whether the receiver class is derived from a given class. For programs that use classes as types, this method can be used to ascertain whether the type of one object is a subtype of another.

This method considers a class object to be descended from itself.

Parameters

receiver	A pointer to the class object to be tested.
aClassObj	A pointer to the potential ancestor class.

Return Value

Returns 1 (true) if *receiver* is derived from *aClassObj,* and 0 (false) otherwise.

C Example

```
#include <dog.h>
/* ---------------------------------------------------
   Note: Dog is a subclass of Animal.
   --------------------------------------------------- */
main()
{
  AnimalNewClass(0,0);
  DogNewClass(0,0);

  if (_somDescendedFrom (_Dog, _Animal))
     somPrintf(”Dog IS descended from Animal\n”);
  else
     somPrintf(”Dog is NOT descended from Animal\n”);
  if (_somDescendedFrom (_Animal, _Dog))
     somPrintf(”Animal IS descended from Dog\n”);
  else
     somPrintf(”Animal is NOT descended from Dog\n”);
```

This program produces the following output:

```
Dog IS descended from Animal
Animal is NOT descended from Dog
```

Original Class

SOMClass

Related Information

Methods: somIsA, **somIsInstanceOf**

somFindMethod, somFindMethodOk Methods

Purpose

Finds the method procedure for a method and indicates whether it represents a static method or a dynamic method. Not generally overridden.

IDL Syntax

boolean somFindMethod (
in somId *methodId***,**
out somMethodPtr *m***);**

boolean somFindMethodOk (
in somId *methodId***,**
out somMethodPtr *m***);**

Note: For backward compatibility, these methods do *not* take an **Environment** parameter.

Description

The **somFindMethod** and **somFindMethodOk** methods perform name-lookup method resolution, determine the method procedure appropriate for performing the indicated method on instances of the receiving class, and load *m* with the method procedure address. For static methods, method procedure resolution is done using the instance method table of the receiving class.

Name-lookup resolution must be used to invoke dynamic methods. Also, name-lookup can be useful when different classes introduce methods of the same name, signature, and desired semantics, but it is not known until runtime which of these classes should be used as a type for the objects on which the method is to be invoked. If the signature of a method is a not known, then method procedures cannot be be used directly, and the **somDispatch** method can be used after dynamically discovering the signature to allow the correct arguments can be placed on a va_list.

As with any methods that return procedure pointers, these methods allow repeated invocations of the same method procedure to be programmed. If this is done, it up to the programmer to prevent runtime errors by assuring that each invocation is performed either on an instance of the class used to resolve the method procedure or of some class derived from it. Whenever using SOM method procedure pointers, it is necessary to indicate the arguments to be passed and the use of system linkage to the compiler, so it can generate a correct procedure call. The way this is done depends on the compiler and the system being used. However, C and C++ usage bindings provide an appropriate typedef for static methods. The name of the typedef is based on the name of the class that introduces the method, as illustrated in the example below.

Unlike the **somFindMethod** method, if the class does not support the specified method, the **somFindMethodOk** method raises an error and halts execution.

If the class does not support the specified method, then ***m** is set to NULL and the return value is meaningless. Otherwise, the returned result is true if the indicated method was a static method.

Parameters

receiver	A pointer to the class object whose method is desired.
methodId	An ID that represents the name of the desired method. The **somIdFromString** function can used to obtain an ID from the method's name.
m	A pointer to the location in memory where a pointer to the specified method's procedure should be stored. Both methods store a NULL pointer in this location (if the method does not exist) or a value that can be called.

Return Value

The **somFindMethod** and **somFindMethodOk** methods return TRUE when the method procedure can be called directly and FALSE when the method procedure is a dispatch function.

C Example

Assuming that the *Animal* class introduces the method *setSound*, the type name for the *setSound* method procedure type will be *somTD_Animal_setSound*, as illustrated below:

```
#include <animal.h>
void main()
{
   Animal myAnimal;
   somId somId_setSound;
   somTD_Animal_setSound methodPtr;
   Environment *ev = somGetGlobalEnvironment();

   myAnimal = AnimalNew();
/* ----------------------------------------
   Note: Next three lines are equivalent to
       _setSound(myAnimal, ev, ”Roar!!!”);
   ---------------------------------------- */
   somId_setSound = somIdFromString(”setSound”);
   _somFindMethod (_somGetClass(myAnimal),
                   somId_setSound, &methodPtr);
   methodPtr(myAnimal, ev, ”Roar!!!”);
/* ---------------------------------------- */
   _display(myAnimal, ev);
   _somFree(myAnimal);
}
/*
Program Output:
This Animal says
Roar!!!
*/
```

Original Class

SOMClass

Related Information

Methods: somFindSMethod, somFindSMethodOk, somSupportsMethod, somDispatch, somClassDispatch

Functions: somApply, somResolve, somClassResolve, somResolveByName, somParentNumResolve

Macros: SOM_Resolve, SOM_ResolveNoCheck, SOM_ParentNumResolve

somFindSMethod, somFindSMethodOk Methods

Purpose

Finds the method procedure for a static method. Not generally overridden.

IDL Syntax

somMethodPtr somFindSMethod (in somId *methodId***);**

somMethodPtr somFindSMethodOk (in somId *methodId***);**

Note: For backward compatibility, these methods do *not* take an **Environment** parameter.

Description

The **somFindSMethod** and **somFindSMethodOk** methods perform name-lookup resolution in a similar fashion to **somFindMethod** and **somFindMethodOk**, but are restricted to static methods. See the description of **somFindMethod** for a discussion of name-lookup method resolution. Because these methods are restricted to resolving static methods, their interface is slightly different from the **somFindMethod** interfaces; a method procedure pointer is returned when lookup is successful; otherwise NULL is returned.

The **somFindSMethodOk** method is identical to **somFindSMethod**, except that an error is raised if the indicated static method is not defined for the receiving class, and execution is halted.

Parameters

receiver	A pointer to a class object.
methodId	A somId representing the name of the desired method.

Return Value

The **somFindSMethod** and **somFindSMethodOk** methods return a pointer to the method procedure that supports the specified method for the class.

Example

See the **somFindMethod** method example.

Original Class

SOMClass

Related Information

Methods: somFindMethod, **somFindMethodOk**

somGetInstancePartSize Method

Purpose

Returns the total size of the instance data structure introduced by a class. Not generally overridden.

IDL Syntax

long somGetInstancePartSize ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetInstancePartSize** method returns the amount of space needed in an object of the specified class or any of its subclasses to contain the instance variables introduced by the class.

Parameters

receiver A pointer to the class object whose instance data size is desired.

Return Value

The **somGetInstancePartSize** method returns the size, in bytes, of the instance variables introduced by this class. This does not include the size of instance variables introduced by this class's ancestor or descendent classes. If a class introduces no instance variables, 0 is returned.

C Example

```
#include <animal.h>
main()
{
  Animal myAnimal;
  SOMClass animalClass;
  int instanceSize;
  int instanceOffset;
  int instancePartSize;

  myAnimal = AnimalNew ();
  animalClass = _somGetClass (myAnimal);
  instanceSize = _somGetInstanceSize (animalClass);
  instanceOffset = _somGetInstanceOffset (animalClass);
  instancePartSize = _somGetInstancePartSize (animalClass);
  somPrintf ("Instance Size: %d\n", instanceSize);
  somPrintf ("Instance Offset: %d\n", instanceOffset);
  somPrintf ("Instance Part Size: %d\n", instancePartSize);
  _somFree (myAnimal);
}
/*
Output from this program:
Instance Size: 8
Instance Offset: 0
Instance Part Size: 4
*/
```

Original Class

SOMClass

Related Information

Methods: somGetInstanceSize

somGetInstanceSize Method

Purpose

Returns the size of an instance of a class. Not generally overridden.

IDL Syntax

long somGetInstanceSize ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetInstanceSize** method returns the total amount of space needed in an instance of the specified class.

Parameters

receiver A pointer to the class object whose instance size is desired.

Return Value

The **somGetInstanceSize** method returns the size, in bytes, of each instance of this class. This includes the space required for instance variables introduced by this class and all of its ancestor classes.

C Example

```
#include <animal.h>
main()
{
  Animal myAnimal;
  SOMClass animalClass;
  int instanceSize;
  int instanceOffset;
  int instancePartSize;

  myAnimal = AnimalNew ();
  animalClass = _somGetClass (myAnimal);
  instanceSize = _somGetInstanceSize (animalClass);
  instanceOffset = _somGetInstanceOffset (animalClass);
  instancePartSize = _somGetInstancePartSize (animalClass);
  somPrintf ("Instance Size: %d\n", instanceSize);
  somPrintf ("Instance Offset: %d\n", instanceOffset);
  somPrintf ("Instance Part Size: %d\n", instancePartSize);
  _somFree (myAnimal);
}
/*
Output from this program:
Instance Size: 8
Instance Offset: 0
Instance Part Size: 4
*/
```

Original Class

SOMClass

Related Information

Methods: somGetInstancePartSize

somGetInstanceToken Method

Purpose

Returns a data access token for the instance data introduced by a class. Not generally overridden.

IDL Syntax

somDToken somGetInstanceToken ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

Returns a data token “pointing” to the beginning of the instance data introduced by the receiving class. This token can be passed to the function **somDataResolve** to locate this instance data within an an instance of the receiver class or any class derived from it. Also the instance data token for a class can be passed to the class method **somGetMemberToken** to get a data token for a specific instance variables introduced by the class (if the relative offset of this instance variable is known). This approach is used by C and C++ implementation bindings to support public instance data for OIDL classes (IDL classes currently have no public instance data).

A data token for the instance data introduced by a class is required by method procedures that access data introduced by the method procedure’s defining class. For classes declared using OIDL and IDL, the needed token is stored in the auxiliary class data structure, which is an external data structure made statically available by the C and C++ language bindings as *<className>*CClassData.instanceToken. Thus, this method call is not generally used by C and C++ class implementors of classes declared using OIDL or IDL.

Parameters

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

Return Value

Returns a data token for the beginning of the instance data introduced by the receiver.

Original Class

SOMClass

Related Information

Functions: somDataResolve

Methods: somGetInstanceSize, somGetInstancePartSize, somGetMemberToken

somGetMemberToken Method

Purpose

Returns an access token for an instance variable. This is method is not generally overridden.

IDL Syntax

somDToken somGetMemberToken (
long *memberOffset*,
somDToken *instanceToken*);

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetMemberToken** method returns an access token for the data member at offset *memberOffset* within the block of instance data identified by *instanceToken*. The returned token can subsequently be passed to the **somDataResolve** function to locate the data member.

Typically, only the code that implements a class declared using OIDL requires access to this method, and this code is normally provided by implementation bindings. Thus C and C++ programmers do not normally invoke this method.

Parameters

receiver	A pointer to a **SOMClass** object.
memberOffset	A 32-bit integer representing the offset of the required data member.
instanceToken	A token, obtained from **somGetInstanceToken**, that identifies the introduced portion of the class.

Return Value

Returns an access token for the specified data member.

Original Class

SOMClass

Related Information

Functions: somDataResolve

Methods: somGetInstanceSize, somGetInstancePartSize, somGetInstanceToken

somGetMethodData Method

Purpose

Returns method information for a specified method, which must have been introduced by the receiver class or an ancestor of that class. Not generally overridden.

IDL Syntax

boolean somGetMethodData (
in somId *methodId***,**
out somMethodData *md***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetMethodData** method loads a *somMethodData* structure with data describing the method identified by the passed *methodId*. If *methodId* does not identify a method known to the receiver, then false is returned; otherwise, true is returned after loading the *somMethodData* structure with data corresponding to the indicated method.

Parameters

receiver	A pointer to the class that produced the index value.
methodId	A **somId** for the method's name.
md	A pointer to a *somMethodData* structure.

Return Value

Boolean true if successful; otherwise false.

C++ Example

```
#include <somcls.xh>
main
{
   somEnvironmentNew();
   somId gmiId = somIdFromString(”somGetMethodIndex”);
   somMethodData md;
   boolean rc = _SOMClass->somGetMethodData(gmiId,&md);
   SOM_Test(rc && somCompareIds(gmiId, md.id));
}
```

Related Information

Data Structures: somMethodData (somapi.h)

Methods: somGetMethodIndex, somGetMethodData, somGetNthMethodInfo

somGetMethodDescriptor Method

Purpose

Returns the method descriptor for a method. Not generally overridden.

IDL Syntax

somId somGetMethodDescriptor (in somId *methodId***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetMethodDescriptor** method returns the method descriptor for a specified method of a class. (A method descriptor is a somId that represents the identifier of an attribute definition or a method definition in the SOM Interface Repository. It contains information about the method's return type and the types of its arguments.) If the class object does not support the indicated method, NULL is returned.

Parameters

receiver	A pointer to a **SOMClass** object.
methodId	A **somId** method descriptor.

Return Value

The **somGetMethodDescriptor** method returns a **somId** method descriptor.

Example

```
somId myMethodDescriptor;
myMethodDescriptor = _somGetMethodDescriptor(_Animal,
                              somIdFromString("setSound"));
```

Original Class

SOMClass

Related Information

Methods: somAddDynamicMethod, somGetNthMethodInfo, somGetMethodData, somGetNthMethodData

somGetMethodIndex Method

Purpose

Returns a class-specific index for a method. Not generally overridden.

IDL Syntax

long somGetMethodIndex (in somId *methodId***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetMethodIndex** method returns an index that can be used in subsequent calls to the same receiving class to determine information about the indicated (static or dynamic) method, via the methods **somGetNthMethodData** and **somGetNthMethodInfo**. The method must be appropriate for use on an instance of the receiver class; otherwise, a –1 is returned. The index of a method can change over time if dynamic methods are added to the receiver class or its ancestors. Thus, in dynamic multi-threaded environments, a critical region should be used to bracket the use of this method and of subsequent requests for method information based on the returned index.

Parameters

receiver	A pointer to a **SOMClass** object.
methodId	A **somId** method ID.

Return Value

The **somGetMethodIndex** method returns a positive long if successful, and a –1 otherwise.

C++ Example

```
#include <somcls.xh>
main
{
   somEnvironmentNew();
   somId gmiId = somIdFromString(”somGetMethodIndex”);
   long index = _SOMClass->somGetMethodIndex(gmiId);
   somMethodData md;
   boolean rc = _SOMClass->somGetNthMethodData(index,&md);
   SOM_Test(rc && somCompareIds(gmiId, md.id));
}
```

Original Class

SOMClass

Related Information

Data Structures: somMethodData (somapi.h)

Methods: somGetNthMethodData, somGetNthMethodInfo

somGetMethodToken Method

Purpose

Returns a method access token for a static method. Not generally overridden.

IDL Syntax

somMToken somGetMethodToken (in somId *methodId***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetMethodToken** method returns a method access token for a static method with the specified ID that was introduced by the receiver class or an ancestor of the receiver class. This method token can be passed to the **somResolve** function (or one of the other offset-based method resolution functions) to select a method procedure pointer from a method table of an object whose class is the same as, or is derived from the class that introduced the method.

Parameters

receiver	A pointer to a **SOMClass** object.
methodId	A **somId** identifying a method.

Return Value

The **somGetMethodToken** method returns a **somMToken** method-access token.

C Example

Assuming that the class *Animal* introduces the method *setSound*,

```
#include <animal.h>
main() {
  somMToken tok;
  Animal myAnimal;
  somTD_Animal_setSound methodPtr; /* use typedef from animal.h */
  Environment *ev = somGetGlobalEnvironment();
  myAnimal = AnimalNew();
  /*next 3 lines equivalent to _setSound(myAnimal, ev, ”Roar!!!”);*/
  tok = _somGetMethodToken(_Animal, somIdFromString(“setSound”));
  methodPtr = (somTD_Animal_setSound)somResolve(myAnimal, tok);
  methodPtr(myAnimal, ev, ”Roar!!!”);
  _display(myAnimal, ev);
  _somFree(myAnimal);
}
```

Original Class

SOMClass

Related Information

Functions: somResolve, somClassResolve, somParentNumResolve

Methods: somGetNthMethodInfo, somGetMethodData

somGetName Method

Purpose

Returns the name of a class. Not generally overridden.

IDL Syntax

string somGetName ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetName** method returns the address of a zero-terminated string that gives the name of the receiving class. This name may be used as a RepositoryId in the **Repository_lookup_id** method (described in the SOM Interface Repository Framework section) to obtain the IDL interface definition that corresponds to the receiving class.

The returned name is not necessarily the same as the statically known class name used by a programmer to gain access to the class object (for example, via the method **somFindClass**). This is because the method **somSubstituteClass** may have been used to “shadow” the class having the static name used by the programmer.

Also, when the interface to a class's instances is defined within an IDL module, the returned name will not directly correspond to the names of the procedures and macros made available by the SOMobjects C and C++ usage bindings for accessing class objects (for example, the **<***className***>NewClass** procedure, or the **_<***className***>** macro). This is because the <*className*> token used in constructing the names of these procedures and macros uses an underscore character to separate the module name from the interface name, while the actual name of the corresponding class uses two colon characters instead of an underscore for this purpose.

The **somGetName** method is not generally overridden. The returned address is valid until the class object is unregistered or freed.

Parameters

receiver The class whose name is desired.

Return Value

The **somGetName** method returns a pointer to the name of the class.

C++ Example

```
#include <animal.xh> /* assume Animal defined in the Zoo module */
#include <string.h>
main()
{
  string className = Zoo_AnimalNewClass(0,0)->somGetName();
  SOM_Test(!strcmp(className, ”Zoo::Animal”));
}
```

Original Class

SOMClass

Related Information

Methods: Repository_lookup_id, **somSubstituteClass, somFindClass**

somGetNthMethodData Method

Purpose

Returns method information for the *n*th (static or dynamic) method known to a given class. Not generally overridden.

IDL Syntax

boolean somGetNthMethodData (
in long ***index,***
out somMethodData *md***)**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetNthMethodData** method loads a *somMethodData* structure with data describing the method identified by the passed index. The index must have been produced by a previous call to exactly the same receiver class; the same method will in general have different indexes in different classes. If the index does not identify a method known to this class, then false is returned; otherwise, true is returned after loading the *somMethodData* structure with data corresponding to the indicated method.

Parameters

receiver	A pointer to the class that produced the index value.
index	An index returned as a result of a previous call of **somGetMethodIndex**.
md	A pointer to a *somMethodData* structure.

Return Value

Boolean true if successful; otherwise, false.

C++ Example

```
#include <somcls.xh>
main
{
   somEnvironmentNew();
   somId gmiId = somIdFromString("somGetMethodIndex");
   long index = _SOMClass->somGetMethodIndex(gmiId);
   somMethodData md;
   boolean rc = _SOMClass->somGetNthMethodData(index,&md);
   SOM_Test(rc && somCompareIds(gmiId, md.id));
}
```

Related Information

Data Structures: somMethodData (somapi.h)

Methods: somGetMethodIndex, somGetMethodData, somGetNthMethodInfo

somGetNthMethodInfo Method

Purpose

Returns the **somId** of the *n*th (static or dynamic) method known to a given class. Also loads a **somId** with a descriptor for the method. Not generally overridden.

IDL Syntax

somId somGetNthMethodInfo (
in long *index,*
out somId *descriptor***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetNthMethodInfo** method returns the identifier of a method, and loads the **somId** whose address is passed with the **somId** of the method descriptor. Method descriptors are used to support access to information stored in a SOM Interface Repository.

Parameters

receiver	A pointer to the class from which the *index* was obtained using method **somGetMethodIndex**.
index	The *n*th method known to this class, whose method descriptor is desired.
descriptor	A pointer to a **somId** that will be loaded with a **somId** for the descriptor.

Return Value

The **somId** for the indicated method, if a method with the indicated index is known to the receiver; otherwise, NULL.

C++ Example

```
#include <somcls.xh>
main()
{
   somEnvironmentNew();
   somId descriptor, icId = somIdFromString("somInitClass");
   long ndx = _SOMClass->somGetMethodIndex(icId);
   SOM_Test(
      somCompareIds(
        icId,
        _SOMClass->somGetNthMethodInfo(ndx,&descriptor));
   SOMFree(icId);
   SOMFree(descriptor);
}
```

Original Class

SOMClass

Related Information

Classes: Repository (repostry.idl)

Methods: somGetMethodIndex, somGetNthMethodData

somGetNumMethods Method

Purpose

Returns the number of methods available for a class. Not generally overridden.

IDL Syntax

long somGetNumMethods ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetNumMethods** method returns the number of methods currently supported by the specified class, including inherited methods (both static and dynamic).

The value that the **somGetNumMethods** method returns is the total number of methods currently known to the receiving class as being applicable to its instances. This includes both static and dynamic methods, whether defined in this class or inherited from an ancestor class.

Parameters

receiver A pointer to the class whose instance method count is desired.

Return Value

The **somGetNumMethods** method returns the total number of methods that are currently available for the receiving class.

C Example

```
#include <animal.h>
main()
{
    int numMethods;

    AnimalNewClass(Animal_MajorVersion, Animal_MinorVersion);
    numMethods = _somGetNumMethods(_Animal);
    somPrintf("Number of methods supported by class: %d\n",
                                                   numMethods);
}
```

Original Class

SOMClass

Related Information

Methods: somGetNumStaticMethods

somGetNumStaticMethods Method

Purpose

Obtains the number of static methods available for a class. Not generally overridden.

IDL Syntax

long somGetNumStaticMethods ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetNumStaticMethods** method returns the number of static methods available in the specified class, including inherited ones. Static methods are those that are represented by entries in the class's instance method table, and which can be invoked using method tokens and offset resolution.

Parameters

receiver A pointer to the class whose static method count is desired.

Return Value

The **somGetNumStaticMethods** method returns the total number of static methods that are available for instances of the receiving class.

C Example

```
#include <animal.h>
main()
{
    long numMethods;

    AnimalNewClass(Animal_MajorVersion, Animal_MinorVersion);
    numMethods = _somGetNumStaticMethods(_Animal);
    somPrintf("Number of static methods supported by class: %d\n",
                 numMethods);
}
```

Original Class

SOMClass

Related Information

Methods: somGetNumMethods

somGetParents Method

Purpose

Gets a pointer to a class's parent (direct base) classes. Not generally overridden.

IDL Syntax

SOMClassSequence somGetParents ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetParents** method returns a sequence containing pointers to the parents of the receiver.

Parameters

receiver A pointer to the class whose parent (base) classes are desired.

Return Value

The **somGetParents** method returns a sequence of pointers to the parents of the receiver, or NULL otherwise (in the case of **SOMObject**). The sequence of parents is in left-to-right order.

C Example

```
/* Note: Dog is a single-inheritance subclass of Animal. */
#include <dog.h>
main()
{
  Dog myDog;
  SOMClass dogClass;
  SOMClassSequence parents;
  char *parentName;
  int i;

  myDog = DogNew();
  dogClass = _somGetClass(myDog);
  parents = _somGetParents(dogClass);
  for (i=0; i<parents._length; i++)
     somPrintf("-- parent %d is %s\n", i,
               _somGetName(parents._buffer[i]));
  _somFree(myDog);
}
/*
Output from this program:
-- parent 0 is Animal
*/
```

Original Class

SOMClass

Related Information

Methods: somGetClass

somGetVersionNumbers Method

Purpose

Gets the major and minor version numbers of a class's implementation code. Not generally overridden.

IDL Syntax

void somGetVersionNumbers (
out long *majorVersion,*
out long *minorVersion***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetVersionNumbers** method returns, via its output parameters, the major and minor version numbers of the class specified by *receiver*. The class object must have already been created (because the class object is the receiver of the method).

Parameters

receiver	A pointer to a class object.
majorVersion	A pointer where the major version number is to be stored.
minorVersion	A pointer where the minor version number is to be stored.

Return Value

None.

C Example

```
#include <som.h>

main() {

  long major, minor;
  SOMClass myClass;

  somEnvironmentNew();
  myClass = _somFindClass(SOMClassMgrObject,
                      somIdFromString("Animal"), 0, 0);
  _somGetVersionNumbers(myClass, &major, &minor);
  somPrintf("The version numbers are %i and %i.\n", major, minor);
}
```

Original Class

SOMClass

Related Information

Methods: somCheckVersion

somLookupMethod Method

Purpose

Performs name-lookup method resolution. Not generally overridden.

IDL Syntax

somMethodPtr somLookupMethod (in somId *methodId***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somLookupMethod** method uses name-lookup resolution to return the address of the method procedure that supports the indicated method on instances of the receiver class. The method may be either static or dynamic. The SOM C and C++ usage bindings support name-lookup method resolution by invoking **somLookupMethod** on the class of the object on which a name-lookup method invocation is made.

The **somLookupMethod** method is like **somFindSMethodOK** except that dynamic methods can also be returned. If the method is not supported by the receiving class, then an error is returned and execution is halted. To check the existence of a method, **somFindMethod** can be used.

As always, in order to use a method procedure pointer such as that returned by **somLookupMethod**, it is necessary to typecast the procedure pointer so that the compiler can create the correct procedure call. This means that a programmer making explicit use of this method must either know the signature of the identified method, and from this create a typedef indicating system linkage and the appropriate argument and return types, or make use of an existing typedef provided by C or C++ usage bindings for a SOM class that introduces a static method with the desired signature.

Parameters

receiver	A pointer to the class whose instance method for the indicated method is desired.
methodId	A **somId** of the method whose method-procedure pointer is needed.

Return Value

A pointer to the method procedure that supports the method indicated by *methodId.* Or, if the method is not supported by the receiving class, then an error is returned, and execution is halted.

C++ Example

```
#include <somcls.xh>
#include <somcm.xh>
void main()
{
   somId fcpId = somIdFromString("somFindClass")
   somId animalId = somIdFromString("Animal");
   SOMClassMgr *cm = somEnvironmentNew();
   somTD_SOMClassMgr_somFindClass findclassproc =
         (somTD_SOMClassMgr_somFindClass)
            _SOMClassMgr->somLookupMethod(fcpId);
   SOMClass *aCls = findclassproc(cm,animalId,0,0);
   ...
   somFree(fcpId);
   somFree(animalId);
 }
```

Original Class

SOMClass

Related Information

Methods: somFindSMethod, somFindSMethodOk, somFindMethod, somFindMethodOk

somNew, somNewNoInit Methods

Purpose

Creates a new instance of a class.

IDL Syntax

SOMObject somNew ( );

SOMObject somNewNoInit ( );

Note: For backward compatibility, these methods do *not* take an **Environment** parameter.

Description

The **somNew** and **somNewNoInit** methods create a new instance of the receiving class. Space is allocated as necessary to hold the new object.

When either of these methods is applied to a class, the result is a new instance of that class. If the receiver class is **SOMClass** or a class derived from **SOMClass**, the new object will be a class object; otherwise, the new object will not be a class object. The **somNew** method invokes the **somDefaultInit** method on the newly created object. The **somNewNoInit** method does not.

Either method can fail to allocate enough memory to hold a new object and, if so, NULL is returned.

The SOM Compiler generates convenience macros for creating instances of each class, for use by C and C++ programmers. These macros can be used in place of this method.

Parameters

receiver A pointer to the class object that is to create a new instance.

Return Value

A pointer to the newly created **SOMObject** object, or NULL.

Example

```
#include <animal.h>

void main()
{    Animal myAnimal;
/* ---------------------------------------------------
Note: next 2 lines are functionally equivalent to
        myAnimal = AnimalNew();
--------------------------------------------------- */
    /* Create class object:. */
    AnimalNewClass(Animal_MajorVersion, AnimalMinorVersion);
    myAnimal = _somNew(_Animal);  /* Create instance of Animal cls */
    /* ... */
    _somFree(myAnimal);       /* Free instance of Animal */
 }
```

Original Class

SOMClass

Related Information

Methods: somRenew

somRenew, somRenewNoInit, somRenewNoInitNoZero, somRenewNoZero Methods

Purpose

Creates a new object instance using a passed block of storage.

IDL Syntax

SOMObject somRenew (in somToken *memPtr***);**

SOMObject somRenewNoInit (in somToken *memPtr***);**

SOMObject somRenewNoInitNoZero (in somToken *memPtr***);**

SOMObject somRenewNoZero (in somToken *memPtr***);**

Note: For backward compatibility, these methods do *not* take an **Environment** parameter.

Description

The **somRenew** method creates a new instance of the receiving class by setting the appropriate location in the passed memory block to the receiving class's instance method table. Unlike **somNew**, these "Renew" methods use the space pointed to by *memPtr* rather than allocating new space for the object. The **somRenew** method automatically re-initializes the object by first zeroing the object's memory, and then invoking **somDefaultInit**; **somRenewNoInit** zeros memory, but does not invoke **somDefaultInit**. **somRenewNoInitNoZero** only sets the method table pointer; while **somRenewNoZero** calls **somDefaultInit**, but does not zero memory first.

No check is made to ensure that the passed pointer addresses enough space to hold an instance of the receiving class. The caller can determine the amount of space necessary by using the **somGetInstanceSize** method.

The C bindings produced by the SOM Compiler contain a macro that is a convenient shorthand for **_somRenew(_***className***)**.

Parameters

receiver	A pointer to the class object that is to create the new instance.
memPtr	A pointer to the space to be used to construct a new object.

Return Value

The value of *newObject* is returned, which is now a pointer to a valid, initialized object.

Example

```
#include <animal.h>

main()
{
  void *myAnimalCluster;
  Animal animals[5];
  SOMClass animalClass;
  int animalSize, i;

  animalClass =
      AnimalNewClass(Animal_MajorVersion,Animal_MinorVersion);
  animalSize = _somGetInstanceSize (animalClass);
  /* Round up to double-word multiple */
  animalSize = ((animalSize+3)/4)*4;
  /*
   * Next line allocates room for 5 objects
   * in a &odq.cluster" with a single memory-
   * allocation operation.
   */
  myAnimalCluster = SOMMalloc (5*animalSize);
  /*
   * The for-loop that follows creates 5 initialized
   * Animal instances within the memory cluster.
   */
  for (i=0; i<5; i++)
     animals[i] =
      _somRenew(animalClass, myAnimalCluster+(i*animalSize));
  /* Uninitialize the animals explicitly: */
  for (i=0; i<5; i++)
     _somUninit(animals[i]);
  /*
   * Finally, the next line frees all 5 animals
   * with one operation.
   */
  SOMFree (myAnimalCluster);
}
```

Original Class

SOMClass

Related Information

Methods: somGetInstanceSize, **somDefaultInit**, **somNew**

somSupportsMethod Method

Purpose

Returns a **boolean** indicating whether instances of a class support a given (static or dynamic) method.

IDL Syntax

boolean somSupportsMethod (in somId *methodId***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somSupportsMethod** method determines if instances of the specified class support the specified (static or dynamic) method.

Parameters

receiver	A pointer to the class object to be tested.
methodId	An ID that represents the name of the method.

Return Value

The **somSupportsMethod** method returns 1 (true) if instances of the specified class support the specified method, and 0 (false) otherwise.

Example

```
/* ----------------------------------------------
   Note: animal supports a setSound method;
         animal does not support a doTrick method.
   ---------------------------------------------- */
#include <animal.h>
main()
{
  SOMClass animalClass;
  char *methodName1 = "setSound";
  char *methodName2 = "doTrick";
  animalClass =
      AnimalNewClass(Animal_MajorVersion, Animal_MinorVersion);
  if (_somSupportsMethod(animalClass,
                         somIdFromString(methodName1)))
     somPrintf("Animals respond to %s\n", methodName1);
  if (_somSupportsMethod(animalClass,
                         somIdFromString(methodName2)))
     somPrintf("Animals respond to %s\n", methodName2);
}

/*
Output from this program:
Animals respond to setSound
*/
```

Original Class

SOMClass

Related Information

Methods: somRespondsTo

SOMClassMgr Class

Description

One instance of **SOMClassMgr** is created automatically during SOM initialization. This instance (pointed to by the global variable, **SOMClassMgrObject**) acts as a run-time registry for all SOM class objects that exist within the current process and assists in the dynamic loading and unloading of class libraries.

You can subclass **SOMClassMgr** to augment the functionality of its registry. To have an instance of your subclass replace the SOM-supplied **SOMClassMgrObject**, use the **somMergeInto** method to place the existing registry information from **SOMClassMgrObject** into your new class-manager object.

File Stem

somcm

Base

SOMObject

Metaclass

SOMClass

Ancestor Classes

SOMObject

Types

interface Repository;
SOMClass *SOMClassArray;

Attributes

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

somInterfaceRepository (Repository)
The SOM Interface Repository object. If the Interface Repository is not available or cannot be initialized, this attribute returns NULL. The object reference returned by this attribute is owned by the **SOMClassMgr** and should not be freed.

somRegisteredClasses (sequence<SOMClass>)
This is a "readonly" attribute that returns a sequence containing all of the class objects registered in the current process. When you have finished using the returned sequence, you should free the sequence's buffer using **SOMFree**. Here is a fragment of code written in C that illustrates the proper use of this attribute:

```
sequence(SOMClass) clsList;

clsList = SOMClassMgr__get_somRegisteredClasses (SOMClassMgrObject);
somPrintf ("Currently registered classes:\n");
for (i=0; i<clsList._length; i++)
    somPrintf ("\t%s\n", SOMClass_somGetName (clsList._buffer[i]));
SOMFree (clsList._buffer);
```

New Methods

Group: Basic Functions

somLoadClassFile
somLocateClassFile
somRegisterClass
somUnloadClassFile
somUnregisterClass

Group: Access

somGetInitFunction
somGetRelatedClasses

Group: Dynamic

somClassFromId
somFindClass
somFindClsInFile
somMergeInto
somSubstituteClass

Overridden Methods

somDumpSelfInt
somInit
somUninit

somClassFromId Method

Purpose

Finds a class object, given its **somId**, if it already exists. Does not load the class.

IDL Syntax

SOMClass somClassFromId (in somId *classId***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

Finds a class object, given its **somId**, if it already exists. Does not load the class.

Use the **somClassFromId** method instead of **somFindClass** when you do *not* want the class to be automatically loaded if it does not already exist in the current process.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
classId	The **somId** of the class. This can be obtained from the name of the class using the **somIdFromString** function.

Return Value

Returns a pointer to the class, or NULL if the class object does not yet exist.

C Example

```
#include <som.h>

main () {
     SOMClass myClass;
     char *myClassName = ”Animal”;
     somId animalId;

     somEnvironmentNew ();
     animalId = somIdFromString (myClassName);
     myClass = SOMClassMgr_somClassFromId (SOMClassMgrObject,
                                                           animalId);
     if (!myClass)
          somPrintf (”Class %s has not been loaded.\n”, myClassName);
     SOMFree (animalId);
     }
```

This program produces the following output:

```
Class Animal has not yet been loaded.
```

Original Class

SOMClassMgr

Related Information

Methods: somFindClass, **somFindClsInFile**

somFindClass Method

Purpose

Finds the class object for a class.

IDL Syntax

SOMClass somFindClass (
in somId *classId***,**
in long *majorVersion***,**
in long *minorVersion***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somFindClass** method returns the class object for the specified class. This method first uses **somLocateClassFile** (see paragraph below) to obtain the name of the file where the class's code resides, then uses **somFindClsInFile**.

If the requested class has not yet been created, the **somFindClass** method attempts to load the class dynamically by loading its dynamically linked library and invoking its "new class" procedure.

The **somLocateClassFile** method uses the following steps: (1) If the entry in the Interface Repository for the class specified by *classId* contains a **dllname** modifier, this value is used as the file name for loading the library. (For information about the **dllname** modifier, refer to the topic "Modifier statements" in Chapter 4, "SOM IDL and the SOM Compiler," of the *SOMobjects Developer Toolkit Users Guide.*) (2) In the absence of a **dllname** modifier, the class name is assumed to be the file name for the library. Use the **somFindClsInFile** method if you wish to explicitly pass the file name as an argument.

If *majorVersion* and *minorVersion* are not both zero, they are used to check the class version information against the caller's expectations. An implementation is compatible with the specified version numbers if it has the same major version number and a minor version number that is equal to or greater than *minorVersion*.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
classId	The **somId** representing the name of the class.
majorVersion	The class's major version number.
minorVersion	The class's minor version number.

Return Values

A pointer to the requested class object, or NULL if the class could not be found or created.

C Example

```
#include <som.h>

/*
 *  This program creates a class object
 *  (from a DLL) without requiring the
 *  usage binding file (.h or .xh) for
 *  the class.
 */

void main ()
{
    SOMClass myClass;
    somId animalId;

    somEnvironmentNew ();
    animalId = somIdFromString ("Animal");

/*  The next statement is equivalent to:
 *    #include "animal.h"
 *    myClass = AnimalNewClass (0, 0);
 */
    myClass = SOMClassMgr_somFindClass (SOMClassMgrObject,
                                         animalId, 0, 0);
    if (myClass)
        somPrintf ("myClass: %s\n", SOMClass_somGetName (myClass));
    else
        somPrintf ("Class %s could not be dynamically loaded\n",
                                 somStringFromId (animalId));
    SOMFree (animalId);
}
```

This program produces the following output:

```
myClass: Animal
```

Original Class

SOMClassMgr

Related Information

Methods: somFindClsInFile, **somLocateClassFile**

somFindClsInFile Method

Purpose

Finds the class object for a class, given a filename that can be used for dynamic loading.

IDL Syntax

SOMClass somFindClsInFile (
in somId *classId*,
in long *majorVersion*,
in long *minorVersion*,
in string *file*);

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somFindClsInFile** method returns the class object for the specified class. This method is the same as **somFindClass** except that the caller provides the filename to be used if dynamic loading is needed.

If the requested class has not yet been created, the **somFindClsInFile** method attempts to load the class dynamically by loading the specified library and invoking its "new class" procedure.

If *majorVersion* and *minorVersion* are not both zero, they are used to check the class version information against the caller's expectations. An implementation is compatible with the specified version numbers if it has the same major version number and a minor version number that is equal to or greater than *minorVersion*.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
classId	The somId representing the name of the class.
majorVersion	The class's major version number.
minorVersion	The class's minor version number.
file	A string representing the filename to be used if dynamic loading is required.

Return Value

A pointer to the requested class object, or NULL if the class could not be found or created.

C Example

```
#include <som.h>
/*
 *  This program loads a class and creates
 *  an instance of it without requiring the
 *  binding (.h) file for the class.
 *
 */
void main()
{
   SOMObject myAnimal;
   SOMClass animalClass;
   char *animalName = "Animal";
        /*
         * Filenames will be different for AIX, OS/2 and Windows
         *
         * Set animalfile to "C:\\MYDLLS\\ANIMAL.DLL" for OS/2
         *                                           or Windows.
         * Set animalfile to "/mydlls/animal.dll" for AIX.
         *
         */

   char *animalFile = "/mydlls/animal.dll";  /* AIX filename */

   somEnvironmentNew();
   animalClass = _somFindClsInFile (SOMClassMgrObject,
                                    somIdFromString(animalName),
                                    0, 0,
                                    animalFile);
   myAnimal = _somNew (animalClass);
   somPrintf("The class of myAnimal is %s.\n",
       _somGetClassName(myAnimal));
   _somFree(myAnimal);
}
/*
Output from this program:
The class of myAnimal is Animal.
*/
```

Original Class

SOMClassMgr

Related Information

Methods: somFindClass

somGetInitFunction Method

Purpose

Obtains the name of the function that initializes the SOM classes in a class library.

IDL Syntax

string somGetInitFunction ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetInitFunction** method supplies the name of the initialization function for OS/2 class libraries (DLLs) that contain more than one SOM class. The default implementation returns the value of the global variable **SOMClassInitFuncName**, which by default is set to the value “SOMInitModule”.

For AIX, the name of the class initialization function is not important, since AIX class libraries should always be constructed as shared libraries with a designated entry point which can be executed automatically by the loader when the class is loaded. Consequently, the result of this method is not significant on AIX.

Similarly, if an OS/2 class library (DLL) has been constructed with a DLL initialization function assigned by the linker, you can choose to invoke the <*className*>**NewClass** functions for all of the classes in the DLL during DLL initialization. In this case (as on AIX), there is no need to export a “SOMInitModule” function. On the other hand, if your compiler does not provide a convenient mechanism for creating a DLL initialization function, you can elect to export a function named “SOMInitModule” (or whatever name is ultimately returned by the **somGetInitFunction** method).

The OS/2 **SOMClassMgrObject**, after loading a class library, will invoke the method **somGetInitFunction** to obtain the name of a possible initialization function. If this name has been exported by the class library just loaded, the **SOMClassMgrObject** calls this function to initialize the classes in the library. If the name has not been exported by the DLL, the **SOMClassMgrObject** then looks for an exported name of the form <*className*>**NewClass**, where <*className*> is the name of the class supplied with the method that caused the DLL to be loaded. If the DLL exports this name, it is invoked to create the named class.

On Windows, the SOM class manager does *not* call **SOMInitModule**. It must be called from the default Windows DLL initialization function, LibMain. This call is made indirectly through the **SOM_ClassLibrary** macro.

Regardless of the technique employed, the **SOMClassMgrObject** expects that all classes packaged in a single class library will be created during this sequence.

This method is generally not invoked directly by users. User-defined subclasses of **SOMClassMgr**, however, can override this method.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).

Return Value

The **somGetInitFunction** method returns a string that names the initialization function of class libraries. By default, this name is the value of the global variable **SOMClassInitFuncName**, the default value of which is **SOMInitModule**.

Original Class

SOMClassMgr

Related Information

Methods: somFindClass, **somFindClsInFile**

Functions: SOMInitModule

Macros: SOM_ClassLibrary

somGetRelatedClasses Method

Purpose

Returns an array of class objects that were all registered during the dynamic loading of a class.

IDL Syntax

SOMClass * somGetRelatedClasses (in SOMClass *classObj***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetRelatedClasses** method returns an array of class objects that were all registered during the dynamic loading of the specified class. These classes are considered to define an affinity group. Any class is a member of at most one affinity group. The affinity group returned by this call is the one containing the class identified by the *classObj* parameter.

The first element in the array is either the class that caused the group to be loaded, or the special value –1, which means that the class manager is currently in the process of unregistering and deleting the affinity group (only class-manager objects would ever see this value). The remainder of the array consists of pointers to class objects, ordered in reverse chronological sequence to that in which they were originally registered. This list includes the given argument, *classObj*, as one of its elements, as well as the class that caused the group to be loaded (also given by the first element of the array). The array is terminated by a NULL pointer as the last element.

Use **SOMFree** to release the array when it is no longer needed. If the supplied class was not dynamically loaded, it is not a member of any affinity group and NULL is returned.

Parameters

receiver Usually a pointer to **SOMClassMgrObject**, or a pointer to an instance of a user-defined subclass of **SOMClassMgr**.

classObj A pointer to a **SOMClass** object.

Return Value

The **somGetRelatedClasses** method returns a pointer to an array of pointers to class objects, or NULL, if the specified class was not dynamically loaded.

Example

```
#include <som.h>
SOMClass myClass, *relatedClasses;
string className;
long i;

className = SOMClass_somGetName (myClass));
relatedClasses = SOMClassMgr_somGetRelatedClasses
                                (SOMClassMgrObject, myClass);
if (relatedClasses && *relatedClasses) {
    somPrintf (”Class=%s, related classes are: ”, className);
    for (i=1; relatedClasses[i]; i++)
        somPrintf (”%s ”,SOMClass_somGetName (relatedClasses[i]));
    somPrintf (”\n”);
    somPrintf (”Class that caused loading was %s\n”,
        relatedClasses[0] == (SOMClass) –1 ? ”–1” :
            SOMClass_somGetName (relatedClasses[0]));
    SOMFree (relatedClasses);
} else
    somPrintf (”No classes related to %s\n”, className);
```

Original Class

SOMClassMgr

Related Information

Methods: somGetInitFunction

somLoadClassFile Method

Purpose

Dynamically loads a class.

IDL Syntax

SOMClass somLoadClassFile (
in somId *classId***,**
in long *majorVersion***,**
in long *minorVersion***,**
in string *file***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **SOMClassMgr** object uses the **somLoadClassFile** method to load a class dynamically during the execution of **somFindClass** or **somFindClsInFile**. A SOM class object representing the class is expected to be created and registered as a result of this method.

The **somLoadClassFile** method can be overridden to load or create classes dynamically using your own mechanisms. If you simply wish to change the name of the procedure that is called to initialize the classes in a library, override **somGetInitFunction** instead.

This method is provided to permit user-created subclasses of **SOMClassMgr** to handle the loading of classes by overriding this method. Do not invoke this method directly; instead, use **somFindClass** or **somFindClsInFile**.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
classId	The **somId** representing the name of the class to load.
majorVersion	The major version number used to check the compatibility of the class's implementation with the caller's expectations.
minorVersion	The minor version number used to check the compatibility of the class's implementation with the caller's expectations.
file	The name of the dynamically linked library file containing the class. The name can be either a simple, unqualified name (without any extension) or a fully qualified (or path) file name, as appropriate for your operating system. For example, on OS/2, *file* could be `c:\myhome\myapp\basename.dll` or else `basename` (but not `basename.dll`).

Return Value

The **somLoadClassFile** method returns a pointer to the class object, or NULL if the class could not be loaded or the class object could not be created.

Original Class

SOMClassMgr

Related Information

Methods: somFindClass, **somFindClsInFile**, **somGetInitFunction**, **somUnloadClassFile**

somLocateClassFile Method

Purpose

Determines the file that holds a class to be dynamically loaded.

IDL Syntax

```
string  somLocateClassFile (
                    in somId classId,
                    in long majorVersion,
                    in long minorVersion);
```

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **SOMClassMgr** object uses the **somLocateClassFile** method when executing **somFindClass** to obtain the name of a file to use when dynamically loading a class. The default implementation consults the Interface Repository for the value of the *dllname* modifier of the class; if no *dllname* modifier was specified, the method simply returns the class name as the expected filename.

If you override the **somLocateClassFile** method in a user-supplied subclass of **SOMClassMgr,** the name you return can be either a simple, unqualified name without any extension or a fully qualified file name. Generally speaking, you would not invoke this method directly. It is provided to permit customization of subclasses of **SOMClassMgr** through overriding.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
classId	The **somId** representing the name of the class to locate.
majorVersion	The major version number used to check the compatibility of the class's implementation with the caller's expectations.
minorVersion	The minor version number used to check the compatibility of the class's implementation with the caller's expectations.

Return Value

The **somLocateClassFile** method returns the name of the file containing the class.

Original Class

SOMClassMgr

Related Information

Methods: somFindClass, **somFindClsInFile**, **somGetInitFunction**, **somLoadClassFile**, **somUnloadClassFile**

somMergeInto Method

Purpose

Transfers SOM class registry information to another **SOMClassMgr** instance.

IDL Syntax

void somMergeInto (in SOMClassMgr *target***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somMergeInto** method transfers the **SOMClassMgr** registry information from one object to another. The target object is required to be an instance of **SOMClassMgr** or one of its subclasses. At the completion of this operation, the target object can function as a replacement for the receiver. The receiver object (which is then in a newly uninitialized state) is placed in a mode where all methods invoked on it will be delegated to the target object. If the receiving object is the instance pointed to by the global variable **SOMClassMgrObject**, then **SOMClassMgrObject** is reassigned to point to the target object.

Subclasses of **SOMClassMgr** that override the **somMergeInto** method should transfer their section of the class manager object from the target to the receiver, then invoke their parent's **somMergeInto** method as the final step.

Invoke this method only if you are creating your own subclass of **SOMClassMgr**. You can invoke **somMergeInto** from an initializer for your new class manager.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
target	A pointer to another instance of **SOMClassMgr** or one of its subclasses.

Return Value

None.

C++ Example

```
// === IDL For the New Class Manager ===

#include <somcm.idl>

interface NewCM : SOMClassMgr {
     implementation {
          somDefaultInit: override;
    };
};

// === C++ implementation for NewCM ===

#define SOM_Module_merge_Source
#include "merge.xih"
```

```
SOM_Scope void SOMLINK somDefaultInit(NewCM *somSelf,
                                      somInitCtrl* ctrl)

{
    NewCMData *somThis; /* set in BeginInitializer */
    somInitCtrl globalCtrl;
    somBooleanVector myMask;
    NewCMMethodDebug("NewCM","somDefaultInit");
    NewCM_BeginInitializer_somDefaultInit;

    NewCM_Init_SOMClassMgr_somDefaultInit(somSelf, ctrl);

    /*
     * local NewCM initialization code added by programmer
     */

    SOMClassMgrObject->somMergeInto(somSelf);
}

// === C++ test program ===

#include <merge.xh>
main()
{
    NewCM *ncm = new NewCM;
    SOMClassMgrObject->somDumpSelf(0);
}

// === Output from test program ===

{An instance of class NewCM at address 20084388
 1 classIdSpaceSize: 3200
 1 classIdHashTableSize: 397
 1 loadAffinity: 0
 1 nextLoadAffinity: 1
 1 IR Class: 00000000, IR Object: 00000000
 1        -Class-- -Token-- Aff Seq ---Id--- Name
 1 [   0] 20077A48 00000000 000 001 2008260C SOMObject
 1 [   1] 2007FB38 00000000 000 000 200825EC SOMClassMgr
 1 [   2] 20083B08 00000000 000 004 2008436C NewCM
 1 [   3] 20077BD8 00000000 000 002 2008262C SOMClass
 1 [   4] 20082668 00000000 000 003 2008315C
SOMParentDerivedMetaclass
}
```

Original Class

SOMClassMgr

somRegisterClass Method

Purpose

Adds a class object to the SOM run-time class registry.

IDL Syntax

void somRegisterClass (in SOMClass *classObj***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somRegisterClass** method adds a class object to the SOM run-time class registry maintained by **SOMClassMgrObject**.

All SOM run-time class objects should be registered with the **SOMClassMgrObject**. This is done automatically during the execution of the **somClassReady** method as class objects are created.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
classObj	A pointer to the class object to add to the SOM class registry.

Return Value

None.

Original Class

SOMClassMgr

Related Information

Methods: somUnregisterClass

somSubstituteClass Method

Purpose

Causes the **somFindClass**, **somFindClsInFile**, and **somClassFromId** methods to substitute one class for another.

IDL Syntax

long somSubstituteClass (
in string *origClassName***,**
in string *newClassName***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somSubstituteClass** method causes the **somFindClass**, **somFindClsInFile**, and **somClassFromId** methods to return the class named *newClassName* whenever they would normally return the class named *origClassName*. This effectively results in class *newClassName* replacing or substituting for class *origClassName*. For example, the <*origClassName*>**New** macro will subsequently create instances of *newClassName.*

Some restrictions are enforced to ensure that this works well. Both class *origClassName* and class *newClassName* must have been already registered before issuing this method, and *newClassName* must be an immediate child of *origClassName*. In addition (although not enforced), no instances should exist of either class at the time this method is invoked.

A convenience macro (**SOM_SubstituteClass**) is provided for C or C++ users. In one operation, it creates both the old and the new class and then substitutes the new one in place of the old. The use of both the **somSubstituteClass** method and the **SOM_SubstituteClass** macro is illustrated in the example below.

Parameters

receiver
Usually **SOMClassMgrObject** or a pointer to an instance of a user-defined subclass of **SOMClassMgr**.

origClassName
A NULL terminated string containing the old class name.

newClassName
A NULL terminated string containing the new class name.

Return Value

The **somSubstituteClass** method returns a value of zero to indicate success; a non-zero value indicates an error was detected.

C Example

```
#include "student.h"
#include "mystud.h"

/* Macro form */
SOM_SubstituteClass (Student, MyStudent);

/* Direct use of the method, equivalent to
 * the macro form above.
 */
{
SOMClass origClass, replacementClass;

origClass = StudentNewClass (Student_MajorVersion,
                                        Student_MinorVersion);
replacementClass = MyStudentNewClass (MyStudent_MajorVersion,
                                      MyStudent_MinorVersion);
SOMClassMgr_somSubstituteClass (
    SOMClass_somGetName (origClass),
    SOMClass_somGetName (replacementClass));
}
```

Original Class

SOMClassMgr

Related Information

Methods: somClassFromId, **somFindClass**, **somFindClsInFile**, **somMergeInto**

somUnloadClassFile Method

Purpose

Unloads a dynamically loaded class and frees the class's object.

IDL Syntax

long somUnloadClassFile (in SOMClass *class***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somUnregisterClass** method uses the **somUnloadClassFile** method to unload a dynamically loaded class. This releases the class's code and unregisters all classes in the same affinity group. (Use **somGetRelatedClasses** to find out which other classes are in the same affinity group.)

The class object is freed whether or not the class' s shared library could be unloaded. If the class was not registered, an error condition is raised and **SOMError** is invoked. This method is provided to permit user-created subclasses of **SOMClassMgr** to handle the unloading of classes by overriding this method. Do not invoke this method directly; instead, invoke **somUnregisterClass**.

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
class	A pointer to the class to be unloaded.

Return Value

The **somUnloadClassFile** method returns 0 if the class was successfully unloaded; otherwise, it returns a system-specific non-zero error code from either the OS/2 **DosFreeModule** or the AIX **unload** system call or the Windows **FreeLibrary** system call.

Original Class

SOMClassMgr

Related Information

Methods: somLoadClassFile, **somRegisterClass**,
somUnregisterClass, somGetRelatedClasses

somUnregisterClass Method

Purpose

Removes a class object from the SOM run-time class registry.

IDL Syntax

long somUnregisterClass (in SOMClass *class***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somUnregisterClass** method unregisters a SOM class and frees the class object. If the class was dynamically loaded, it is also unloaded using **somUnloadClassFile** (which causes its entire affinity group to be unloaded as well).

Parameters

receiver	Usually **SOMClassMgrObject** (or a pointer to an instance of a user-supplied subclass of **SOMClassMgr**).
class	A pointer to the class to be unregistered.

Return Value

The **somUnregisterClass** method returns 0 for a successful completion, or non-zero to denote failure.

Example

```
#include <som.h>

void main ()
{
    long rc;  /* Return code */
    SOMClass animalClass;

    /* The next 2 lines declare a static form of somId */
    string animalClassName = "Animal";
    somId animalId = &animalClassName;

    somEnvironmentNew ();
    animalClass = SOMClassMgr_somFindClass (SOMClassMgrObject,
                                            animalId, 0, 0);
    if (!animalClass) {
        somPrintf ("Could not load class.\n");
        return;
    }
    rc = SOMClassMgr_somUnregisterClass (SOMClassMgrObject,
                                               animalClass);
    if (rc)
        somPrintf ("Could not unregister class, error code: %ld.\n",
                                                                rc);
    else
        somPrintf ("Class successfully unloaded.\n");
}
```

Original Class

SOMClassMgr

Related Information

Methods: somLoadClassFile, **somRegisterClass**, **somUnloadClassFile**

SOMObject Class

SOMObject is the root class for all SOM classes. That is, all SOM classes must be subclasses of **SOMObject** or of some other class derived from **SOMObject**. **SOMObject** introduces no instance data, so objects whose classes inherit from **SOMObject** incur no size increase. They do inherit a suite of methods that provide the behavior required of all SOM objects. Three of these methods are typically overridden by any subclass that has instance data — **somDefaultInit**, **somDestruct**, and **somDumpSelfInt**. See the descriptions of these methods for more information.

File Stem

somobj

Base

None

Metaclass

SOMClass

Ancestor Classes

None

New Methods

Group: Initialization/Termination

somFree
somDefaultInit
somDestruct
somInit
somUninit
somDefaultAssign
somDefaultConstAssign
somDefaultConstCopyInit
somDefaultCopyInit

Group: Access

somGetClass
somGetClassName
somGetSize

Group: Testing

somIsA
somIsInstanceOf
somRespondsTo

Group: Dynamic

somDispatchA
somDispatchD
somDispatchL
somDispatchV
somDispatch
somClassDispatch
somCastObj
somResetObj

Group: Development Support

somDumpSelf
somDumpSelfInt
somPrintSelf

Overridden Methods

None

somCastObj Method

Purpose

Changes the behavior of an object to that defined by any ancestor of the true class of the object.

IDL Syntax

boolean somCastObj (in SOMClass *ancestor***);**

Description

The **somCastObj** method changes the behavior of an object so that its behavior will be that of an instance of the indicated ancestor class (with respect to any method supported by the ancestor). The behavior of the object on methods not supported by the ancestor remains unchanged.

This operation actually changes the class of the object (since an object's behavior is defined by its class). The name of the new class is derived from the initial name of the object's class and the name of the ancestor class, as illustrated in the example below.

The **somCastObj** method may be used on an object multiple times, always with the restriction that the ancestor class whose behavior is selected is actually an ancestor of the true (original) class of the object.

Parameters

receiver	A pointer to an object of type **SOMObject**.
ancestor	A pointer to a class that is an ancestor of the actual class of the *receiver*.

Return Value

The **somCastObj** method returns 1 (TRUE) if the operation is successful and 0 (false) otherwise. The operation fails if *ancestor* is not actually an ancestor of the class of the object.

Example

```
#include <som.h>
main()
{
   SOMClassMgr cm = somEnvironmentNew();
   SOM_Test(1 == _somCastObj(cm, _SOMObject));
   _somDumpSelf(cm, 0));
   SOM_Test(1 == _somResetObj(cm));
   _somDumpSelf(cm, 0);
}

/* output:
 *  {An instance of class SOMClassMgr->SOMObject
 *   at address 20061268
 *  }
 *  {An instance of class SOMClassMgr at address 20061268
 *   ... <SOMClassMgr State Information> ...
 *  }
 */
```

Original Class

SOMObject

Related Information

Methods: somResetObj

somDefaultAssign Method

Purpose

Provides support for an object-assignment operator. May be overridden, but, if appropriate, **somDefaultConstAssign** should be overridden instead.

IDL Syntax

void somDefaultAssign (inout somInitCtrl *ctrl*,
in SOMObject *fromObj* **);**

Description

In C++, assignment to an object of class "X" is accomplished by using (an appropriate overloading of) the assignment operator provided by "X." To make assignment available on all SOM objects, **SOMObject** provides the **somDefaultAssign** and **somDefaultConstAssign** methods. The default behavior of these methods is that they do a shallow copy of data from one object to another. Users should generally use the **somDefaultAssign** method for doing object assignment.

When a shallow copy is not appropriate for the data introduced by a class, and it is possible to perform the copy without modifying *fromObj*, it is recommended that the class implementor override the **somDefaultConstAssign** method for that class.

The considerations important to overriding **somDefaultAssign** are similar to those described in the *SOMobjects Users Guide* for overriding **somDefaultInit**. (See "Initializing and Uninitializing Objects" in Chapter 5, "Implementing Classes in SOM.") The basic difference between **somDefaultInit** and **somDefaultAssign** is that the latter method takes an object (*fromObj*) as a source argument for assignment of values to the receiver.

Parameters

receiver	A pointer to an object of an arbitrary SOM class, *S*.
ctrl	A pointer to a **somInitCtrl** structure, or NULL.
fromObj	A pointer to an object of class *S* or some class descended from *S*.

Return Value

None.

Example

```
// C++ SOMObjects Toolkit Code
#include <Y.xh>

main()
{
   X *x = new X;
   Y *y = new Y; // assume Y is derived from X
   x->somDefaultAssign(0,y)
   // the x object has now been assigned values from y
}
```

Original Class

SOMObject

Related Information

Methods: somDefaultInit, somDefaultConstAssign, somDefaultCopyInit, somDefaultConstCopyInit

somDefaultConstAssign Method

Purpose

Provides support for a "const" object-assignment operator. Designed to be overridden.

IDL Syntax

void somDefaultConstAssign (inout somInitCtrl *ctrl*,
in SOMObject *fromObj* **);**

Description

In C++, assignment to an object of class "X" is accomplished by using (an appropriate overloading of) the assignment operator provided by "X." To make assignment available on all SOM objects, **SOMObject** introduces the **somDefaultAssign** and **somDefaultConstAssign** methods. The default behavior of these methods is to perform a shallow copy of data from one object to another. When this default is not appropriate for a class, and it is possible to perform the copy without modifying *fromObj*, it is recommended that the class implementor override the **somDefaultConstAssign** method.

Generally, an object user should use the **somDefaultAssign** method to perform object assignment.

The considerations important to overriding **somDefaultConstAssign** are similar to those described in the *SOMobjects Users Guide* for overriding **somDefaultInit**. (See "Initializing and Uninitializing Objects" in Chapter 5, "Implementing Classes in SOM.") The basic difference between **somDefaultInit** and **somDefaultConstAssign** is that the latter method takes an object (*fromObj*) as an argument that is to be copied.

Parameters

receiver	A pointer to an object of an arbitrary SOM class, *S*.
ctrl	A pointer to a **somInitCtrl** structure, or NULL.
fromObj	A pointer to an object of class *S* or some class descended from *S*.

Return Value

None.

Example

```
// IDL for a class that overrides somDefaultConstAssign
#include <x.idl>

interface Y : X {
    implementation {
        somDefaultConstAssign: override, init;
    };
};
```

Original Class

SOMObject

Related Information

Methods: somDefaultInit, somDefaultAssign, somDefaultCopyInit, somDefaultConstCopyInit

somDefaultConstCopyInit Method

Purpose

Provides support for passing objects as call-by-value object parameters in methods introduced by DTS C++ classes. Designed to be overridden.

IDL Syntax

void somDefaultConstCopyInit (inout somInitCtrl *ctrl*,
in SOMObject *fromObj* **);**

Description

The **somDefaultConstCopyInit** method would be called a “copy constructor” in C++. In SOM, this concept is supported using an object initializer that accepts the object to be copied as an argument. Copy constructors are used in C++ to pass objects by value. They initialize one object by making it be a copy of another object. In SOM, objects are always passed by reference, so arguments to DTS C++ methods that receive call-by-value object parameters are actually passed by reference. But, to correctly support the semantics of DTS C++ call-by-value arguments, it is necessary to actually pass a *copy* of the intended argument. A copy constructor can be used to make this copy.

The default behavior provided by **somDefaultConstCopyInit** is to do a shallow copy of each ancestor class’s introduced instance variables. The object being copied is not changed. When a shallow copy is not appropriate, and it is possible to avoid changing *fromObj*, a class implementor should override **somDefaultConstCopyInit** (for example, to do a deep copy for certain variables), but should respect the constraint of not modifying the object being copied.

In general, object users should use **somDefaultCopyInit** to copy an object.

The considerations important to overriding **somDefaultConstCopyInit** are similar to those described in the *SOMobjects Users Guide* for overriding **somDefaultInit**. (See “Initializing and Uninitializing Objects” in Chapter 5, “Implementing Classes in SOM.”) The basic difference between **somDefaultInit** and **somDefaultConstCopyInit** is that the latter method takes an object (*fromObj*) as an argument that is to be copied.

Parameters

receiver	A pointer to an uninitialized object of an arbitrary SOM class, *S*.
ctrl	A pointer to a **somInitCtrl** structure, or NULL.
fromObj	A pointer to an object of class *S* or some class descended from *S*.

Return Value

None.

Example

```
// IDL for a class that overrides somDefaultConstCopyInit
interface X : SOMObject
{
    implementation {
        somDefaultConstCopyInit: override, init;
    };
};
```

Original Class

SOMObject

Related Information

Methods: somDefaultInit, somDefaultCopyInit, somDefaultAssign, somDefaultConstAssign

somDefaultCopyInit Method

Purpose

Provides support for call-by-value object parameters in methods introduced by DTS C++ classes. May be overridden, but, if appropriate, **somDefaultConstCopyInit** should be overridden instead.

IDL Syntax

void somDefaultCopyInit (inout somInitCtrl *ctrl*,
in SOMObject *fromObj* **);**

Description

The **somDefaultCopyInit** method would be called a "copy constructor" in C++. In SOM, this concept is supported using an object initializer that accepts the object to be copied as an argument. Copy constructors are used in C++ to pass objects by value. They initialize one object by making it be a copy of another object. In SOM, objects are always passed by reference, so arguments to DTS C++ methods that receive call-by-value object parameters are actually passed by reference. But, to correctly support the semantics of DTS C++ call-by-value arguments, it is necessary to actually pass a *copy* of the intended argument. In general, **somDefaultCopyInit** should be used to make this copy.

The default behavior provided by **somDefaultCopyInit** is to do a shallow copy of each ancestor class's introduced instance variables. However, a class may always override this default behavior (for example, to do a deep copy for certain variables). If it is possible to avoid modification of *fromObj* when doing the copy, the method **somDefaultConstCopyInit** should be overridden for this purpose. Only if this is not possible (and shallow copy is not appropriate) would it be appropriate to override **somDefaultCopyInit**.

The considerations important to overriding **somDefaultCopyInit** are similar to those described in the *SOMobjects Users Guide* for overriding **somDefaultInit**. (See "Initializing and Uninitializing Objects" in Chapter 5, "Implementing Classes in SOM.") The basic difference between **somDefaultInit** and **somDefaultCopyInit** is that the latter method takes an object (*fromObj*) as an argument that is to be copied.

Parameters

receiver	A pointer to an uninitialized object of an arbitrary SOM class, *S.*
ctrl	A pointer to a **somInitCtrl** structure, or NULL.
fromObj	A pointer to an object of class *S* or some class descended from *S.*

Return Value

None.

Example

```
// IDL produced by a DTS C++ compiler for a DTS C++ class
interface X : SOMObject
{
   void foo(in SOMClass arg);
   implementation {
      foo: cxxdecl = ”void foo(SOMClass arg)”; // !! call-by-value
   };
};

// C++ SOMObjects Toolkit Code
#include <X.xh>
#include <somcls.xh>
main()
{
   X *x = new X;
   SOMClass *arg = _SOMClass->somNewNoInit();
   // make arg be a copy of the X class object
   arg->somDefaultCopyInit(0,_X);
   x->foo(arg); // call foo with the copy
}
```

Original Class

SOMObject

Related Information

Methods: somDefaultInit, somDefaultConstCopyInit, somDefaultAssign, somDefaultConstAssign

somDefaultInit Method

Purpose

Initializes instance variables and attributes in a newly created object. Replaces **somInit** as the preferred method for default object initialization. For performance reasons, it is recommended that **somDefaultInit** always be overridden by classes.

IDL Syntax

void somDefaultInit (inout somInitCtrl *ctrl* **);**

Description

Every SOM class is expected to support a set of initializer methods. This set will always include **somDefaultInit**, whether or not the class explicitly overrides **somDefaultInit**. All other initializer methods for a class must be explicitly introduced by the class. See Section 5.5, "Initializing and Uninitializing Objects," of the *SOMobjects Developer Toolkit Users Guide* for complete information on introducing new initializers.

The purpose of an initializer method supported by a class is first to invoke initializer methods of ancestor classes (those ancestors that are the class's **directinitclasses**) and then to place the instance variables and attributes introduced by the class into some consistent state by loading them with appropriate values. The result is that, when an object is initialized, each class that contributes to its implementation will run some initializer method. The **somDefaultInit** method may or may not be among the initializers used to initialize a given object, but it is always available for this purpose.

Thus, the **somDefaultInit** method may be invoked on a newly created object to initialize its instance variables and attributes. The **somDefaultInit** method is more efficient than **somInit** (the method it replaces), and it also prevents multiple initializer calls to ancestor classes. The **somInit** method is now considered obsolete when writing new code, although **somInit** is still supported.

To override **somDefaultInit**, the **implementation** section of the class's .idl file should include **somDefaultInit** with the **override** and **init** modifiers specified. (The **init** modifier signifies that the method is an *initializer* method.) No additional coding is required for the resulting **somDefaultInit** stub procedure in the implementation template file, unless the class implementor wishes to customize object initialization in some way.

If the .idl file does *not* explicitly override **somDefaultInit**, then by default a generic method procedure for **somDefaultInit** will be provided by the SOMobjects Toolkit. If invoked, this generic method procedure first invokes **somDefaultInit** on the appropriate ancestor classes, and then (for consistency with earlier versions of SOMobjects) calls any **somInit** code that may have been provided by the class (if **somInit** was overridden). Because the generic procedure for **somDefaultInit** is less efficient than the stub procedure that is provided when **somDefaultInit** is overridden, it is recommended that the .idl file always override **somDefaultInit.**

Note: It is *not* appropriate to override both **somDefaultInit** and **somInit**. If this is done, the **somInit** code will not be executed. The best way to convert an old class that overrides **somInit** to use of the more efficient **somDefaultInit** (if this is desired) is as follows: (1) Replace the **somInit** override in the class's .idl file with an override for **somDefaultInit**, (2) run the implementation template emitter to produce a stub procedure for **somDefaultInit**, and then (3) simply call the class's **somInit** procedure directly (*not* using a method invocation) from the **somDefaultInit** method procedure.

As mentioned above, the object-initialization framework supported by SOMobjects allows a class to support additional initializer methods besides **somDefaultInit.** These additional initializers will typically include special-purpose arguments, so that objects of the class can be initialized with special capabilities or characteristics. For each new initializer method, the **implementation** section must include the method name with the **init** modifier. Also, the **directinitclasses** modifier can be used if, for some reason, the class implementor wants to control the order in which ancestor initializers are executed.

Notes: It is recommended that the method name for an *initializer* method include the class name as a prefix. A newly defined initializer method will include an implicit Environment argument if the class does not use a **callstyle=oidl** modifier.

Important: There are important constraints associated with modification of the procedure stubs for initializers. These are documented in Section 5.5 of the *SOMobjects Developer Toolkit Users Guide*.

Parameters

receiver A pointer to an object.

ctrl A pointer to a **somInitCtrl** data structure. SOMobjects uses this data structure to control the initialization of the ancestor classes, thereby ensuring that no ancestor class receives multiple initialization calls.

Return Value

None.

Example

```
// SOM IDL
#include <Animal.idl>

interface Dog : Animal
{
    implementation {
        releaseorder: ;
            somDefaultInit: override, init;
        };
};
```

Original Class

SOMObject

Related Information

Methods: somDestruct

somDestruct Method

Purpose

Uninitializes the receiving object, and (if so directed) frees object storage after uninitialization has been completed. Replaces **somUninit** as the preferred method for uninitializing objects. For performance reasons, it is recommended that **somDestruct** always be overridden. Not normally invoked directly by object clients.

IDL Syntax

void somDestruct (in octet *dofree*, **inout somDestructCtrl** *ctrl* **);**

Description

Every class must support the **somDestruct** method. This is accomplished either by overriding **somDestruct** (in which case a specialized stub procedure will be generated in the implementation template file), or else SOMobjects will automatically provide a generic procedure that implements **somDestruct** for the class. The generic procedure calls **somUninit** (if this was overridden) to perform local uninitialization, then completes execution of the method appropriately.

Because the specialized stub procedure generated by the template emitter is more efficient than the generic procedure provided when **somDestruct** is not overridden, it is recommended that **somDestruct** always be overridden. The stub procedure that is generated in this case requires no modification for correct operation. The only modification appropriate within this stub procedure is to uninitialize locally introduced instance variables. See Section 5.5, "Initializing and Uninitializing Objects," of the *SOMobjects Developer Toolkit Users Guide* for further details.

Uninitialization with **somDestruct** executes as follows: For any given class in the ancestor chain, **somDestruct** first uninitializes that class's introduced instance variables (if this is appropriate), and then calls the next ancestor class's implementation of **somDestruct**, passing 0 (that is, false) as the interim *dofree* argument. Then, after all ancestors of the given class have been uninitialized, if the class's own **somDestruct** method were originally invoked with *dofree* as 1 (that is, true), then that object's storage is released.

Note: It is *not* appropriate to override both **somDestruct** and **somUninit**. If this is done, the **somUninit** code will not be executed. The best way to convert an old class that overrides **somUninit** to use of the more efficient **somDestruct** (if this is desired) is as follows: (1) Replace the **somUninit** override in the class's .idl file with an override for **somDestruct**, (2) run the emitter to produce a stub procedure for **somDestruct** in the implementation template file, and then (3) simply call the class's **somUninit** procedure directly (*not* using a method invocation) from the **somDestruct** procedure.

Parameters

receiver — A pointer to an object**.**

dofree — A boolean indicating whether the caller wants the object storage freed after uninitialization of the current class has been completed. Passing 1 (true) indicates the object storage should be freed.

ctrl — A pointer to a **somDestructCtrl** data structure. SOMobjects uses this data structure to control the uninitialization of the ancestor classes, thereby ensuring that no ancestor class receives multiple uninitialization calls. If a user invokes **somDestruct** on an object directly, a NULL (that is, zero) ctrl pointer can be passed. This instructs the receiving code to obtain a **somDestructCtrl** data structure from the class of the object.

Return Value

None.

Example

```
// SOM IDL
#include <Animal.idl>

interface Dog : Animal
{
    implementation {
        releaseorder: ;
            somDestruct: override;
        };
};
```

Original Class

SOMObject

Related Information

Methods: somDefaultInit

somDispatch, somClassDispatch Methods

Purpose

Invokes a method using dispatch method resolution. The **somDispatch** method is designed to be overridden. The **somClassDispatch** method is not generally overridden.

IDL Syntax

boolean somDispatch (
out somToken *retValue***,**
in somId *methodId***,**
in va_list *args***);**

boolean somClassDispatch (
in SOMClass *clsObj***,**
out somToken *retValue***,**
in somId *methodId***,**
in va_list *args***);**

Note: For backward compatibility, these methods do *not* take an **Environment** parameter.

Description

Both **somDispatch** and **somClassDispatch** perform method resolution to select a method procedure, and then invoke this procedure on *args*. The "somSelf" argument for the selected method procedure (called the "target object," below, to distinguish it from the receiver of the **somDispatch** or **somClassDispatch** method call) is the first argument included in the va_list, *args*.

For **somDispatch**, method resolution is performed using the class of the receiver; for **somClassDispatch**, method resolution is performed using the argument class, *clsObj*. Because **somClassDispatch** uses *clsObj* for method resolution, a programmer invoking **somDispatch** or **somClassDispatch** should assure that the class of the target object is either derived from or is identical to the class used for method resolution; otherwise, a run-time error will likely result when the target object is passed to the resolved procedure. Although not necessary, the receiver is usually also the target object.

The **somDispatch** and **somClassDispatch** methods supersede the **somDispatch***X* methods. Unlike the **somDispatch***X* methods, which are restricted to few return types, the **somDispatch** and **somClassDispatch** methods make no assumptions concerning the result returned by the method to be invoked. Thus, **somDispatch** and **somClassDispatch** can be used to invoke methods that return structures. The **somDispatch***X* methods now invoke **somDispatch**, so overriding **somDispatch** serves to override the **somDispatch***X* methods as well.

Parameters

receiver	A pointer to the object whose class will be used for method resolution by **somDispatch**.
clsObj	A pointer to the class that will be used for method resolution by **somClassDispatch**.
retValue	The address of the area in memory where the result of the invoked method procedure is to be stored. The caller is responsible for allocating enough memory to hold the result of the specified method. When dispatching methods that return no result (that is, void), a NULL may be passed as this argument.
methodId	A **somId** identifying the method to be invoked. A string representing the method name can be converted to a **somId** using the **somIdFromString** function.

args A **va_list** containing the arguments to be passed to the method identified by *methodId*. The arguments must include a pointer to the target object as the first entry. As a convenience for C and C++ programmers, SOM's language bindings provide a varargs invocation macro for va_list methods (such as **somDispatch** and **somClassDispatch).** The example below illustrates this.

Return Value

A boolean representing whether or not the method was successfully dispatched is returned. The reason for this is that **somDispatch** and **somClassDispatch** use the function **somApply** to invoke the resolved method procedure, and **somApply** requires an apply stub for successful execution. In support of old class binaries SOM does not consider a NULL apply stub to be an error. As a result, somApply may fail. If this happens, then false is returned; otherwise, true is returned.

C Example

Given class *Key* that has an attribute *keyval* of type **long** and an overridden method for **somPrintSelf** that prints the value of the attribute (as well as the information printed by **SOMObject**'s implementation of **somPrintSelf**), the following client code invokes methods on *Key* objects using **somDispatch** and **somClassDispatch**. (The *Key* class was defined with the **callstyle=oidl** class modifier, so the **Environment** argument is not required of its methods.)

```
#include <key.h>

main()
{
  SOMObject obj;
  long k1 = 7, k2;
  Key myKey = KeyNew();
  va_list push, args = SOMMalloc(8);
  somId setId = somIdFromString(”_set_keyval”);
  somId getId = somIdFromString(”_get_keyval”);
  somId prtId = somIdFromString(”somPrintSelf”);

  /* va_list invocation of setkey and getkey : */
  push = args;
  va_arg(push, SOMObject) = myKey;
  va_arg(push, long) = k1;
  SOMObject_somDispatch(myKey,(somToken*)0,setId,args);
  push = args;
  va_arg(push, SOMObject) = myKey;
  SOMObject_somDispatch(myKey,(somToken*)&k2,getId,args);
  printf(”va_list _set_keyval and _get_keyval: %i\n”, k2);

  /* varargs invocation of setkey and getkey : */
  _somDispatch(myKey, (somToken*)0, setId, myKey, k1);
  _somDispatch(myKey, (somToken*)&k2, getId, myKey);
  printf(”varargs _set_keyval and _get_keyval: %i\n”, k2);

  /* illustrate somclassDispatch ”casting” (use varargs form) */
  printf(”somPrintSelf on myKey as a Key:\n”);
  _somClassDispatch(myKey,_Key,(somToken*)&obj2,prtId,myKey,0);

  printf(”somPrintSelf on myKey as a SOMObject:\n”);
  _somClassDispatch(myKey,_SOMObject,(somToken*)&obj,prtId,myKey,0);
  SOMFree(args); SOMFree(setId); SOMFree(getId); SOMFree(prtId);
 _somFree(myKey);
}
```

This program produces the following output:

```
va_list _set_keyval and _get_keyval: 7
varargs _set_keyval and _get_keyval: 7
somPrintSelf on myKey as a Key:
{An instance of class Key at address 2005B2F8}
    -- with key value 7
somPrintSelf on myKey as a SOMObject:
{An instance of class Key at address 2005B2F8}
```

Original Class

SOMObject

Related Information

Functions: somApply

somDispatch*X* Methods *(Obsolete)*

Purpose

Invoke a method using dispatch method resolution. These methods are obsolete.

IDL Syntax

somToken somDispatchA (
in somId *methodId***,**
in somId *descriptor***,**
in va_list *args***);**

double somDispatchD (
in somId *methodId***,**
in somId *descriptor***,**
in va_list *args***);**

long somDispatchL (
in somId *methodId***,**
in somId *descriptor***,**
in va_list *args***);**

void somDispatchV (
in somId *methodId***,**
in somId *descriptor***,**
in va_list *args***);**

Note: For backward compatibility, these methods do *not* take an **Environment** parameter.

Description

The **somDispatch*X*** methods are superseded by the more general **somDispatch** method, and are retained solely for backward compatibility.

The **somDispatch*X*** methods invoke on the receiving object the method identified by *methodId*, with arguments specified by *args*. The target object for the method invocation is the receiving object, which is *not* included in the arguments.

Parameters

receiver	A pointer to the object on which the dispatched method is invoked.
methodId	A **somId** that represents the method to be invoked.
descriptor	A **somId** that represents the types of the arguments being passed in the *args* **va_list**. This parameter is not used in the current implementation, so a NULL value can be substituted.
args	A **va_list** containing the arguments to be passed to the method identified by *methodId*. The arguments do not include the target for the dispatched method.

Return Value

Four families of return values are supported, corresponding to the four forms of the **somDispatch***X* method. The **somDispatch***X* method chosen should have a return type compatible with the result of the method identified by *methodId*. Within each of the four families, only the largest representation is supported. The four families are:

Pointer	**somDispatchA** returns an address as a **somToken**.
Floating point	**somDispatchD** returns a floating point number as a **double.**
Integer	**somDispatchL** returns an integer as a **long**.
Void	**somDispatchV** returns **void**. It is used for methods that do not return a result.

Original Class

SOMObject

Related Information

Functions: somApply

Methods: somDispatch

somDumpSelf Method

Purpose

Writes out a detailed description of the receiving object. Intended for use by object clients. Not generally overridden.

IDL Syntax

void somDumpSelf (in long *level***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somDumpSelf** method performs some initial setup, and then invokes the **somDumpSelfInt** method to write a detailed description of the receiver, including its state.

Parameters

receiver	A pointer to the object to be dumped.
level	The nesting level for describing compound objects. It must be greater than or equal to 0. All lines in the description will be preceded by "2 * level" spaces.

Return Value

None.

Example

See **somDumpSelfInt.**

Original Class

SOMObject

Related Information

Methods: somDumpSelfInt

somDumpSelfInt Method

Purpose

Outputs the internal state of an object. Intended to be overridden by class implementors. Not intended to be directly invoked by object clients.

IDL Syntax

void somDumpSelfInt (in long *level***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somDumpSelfInt** method should be overridden by a class implementor, to write out the instance data stored in an object. This method is invoked by the **somDumpSelf** method, which is used by object clients to output the state of an object.

The procedure used to override this method for a new class should begin by calling the parent class form of this method on each of the class parents, and should then write a description of the instance variables introduced by new class. This will result in a description of all the class's instance variables. The C and C++ implementation bindings provide a convenient macro for performing parent method calls on all parents, as illustrated below.

The character output routine pointed to by **SOMOutCharRoutine** should be used for output. The **somLPrintf** function is especially convenient for this, since level is handled appropriately.

Parameters

receiver A pointer to the object to be dumped.

level The nesting level for describing compound objects. It must be greater than or equal to 0. All lines in the description should be preceded by "2 * level" spaces.

Return Value

None.

C Example

Below is a method overriding **somDumpSelfInt** for class "List", which has two attributes, *val* (which is a **long**) and *next* (which is a pointer to a "List" object).

```
SOM_Scope void   SOMLINK somDumpSelfInt(List somSelf, int level)
{
    ListData *somThis = ListGetData(somSelf);
    Environment *ev = somGetGlobalEnvironment();

    List_parents_somDumpSelfInt(somSelf, level);
    somLPrintf(level, "This item: %i\n", __get_val(somSelf, ev);
    somLPrintf(level, "Next item: \n");
    if (__get_next(somSelf, ev) != (List) NULL)
        _somDumpSelfInt(__get_next(somSelf, ev), level+1);
    else
        somLPrintf(level+1, "NULL\n");
}
```

Below is a client program that invokes the **somDumpSelf** method on "List" objects:

```
#include <list.h>

main()
{
   List L1, L2;
   long x = 7, y = 13;
   Environment *ev = somGetGlobalEnvironment();

   L1 = ListNew();
   L2 = ListNew();
   __set_val(L1, ev, x);
   __set_next(L1, ev, (List) NULL);
   __set_val(L2, ev, y);
   __set_next(L2, ev, L1);

   _somDumpSelf(L2,0);

   _somFree(L1);
   _somFree(L2);
}
```

Below is the output produced by this program:

```
{An instance of class List at 0x2005EA8
 This item: 13
 Next item:
   1 This item: 7
   1 Next item:
     2 NULL
}
```

Original Class

SOMObject

Related Information

Methods: somDumpSelf, **somPrintSelf**

somFree Method

Purpose

Releases the storage used by an object and frees the object. Intended for use by object clients. Not generally overridden.

IDL Syntax

void somFree ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somFree** method releases the storage containing the receiver object by calling the method **somDeallocate**. No future references should be made to the receiver once this is done. The **somFree** method calls **somDestruct** to allow storage pointed to by the object to be freed.

The **somFree** method should not be called on objects created by **somRenew**, thus the method is normally only used by code that also created the object.

Note: SOM also supplies a function, **SOMFree**, which is used to free a block of memory. This function should not be used on objects.

Parameters

receiver A pointer to the object to be freed.

Return Value

None.

C Example

```
#include <animal.h>

void main()
{
   Animal myAnimal;
   /*
    * Create an object.
    */
   myAnimal = AnimalNew();

   /* ... */

   /* Free it when finished. */
   _somFree(myAnimal);
}
```

Original Class

SOMObject

Related Information

Methods: somNew, **somNewNoInit**, **somDestruct**

Functions: SOMFree

somGetClass Method

Purpose

Returns a pointer to an object's class object. Not generally overridden.

IDL Syntax

SOMClass somGetClass ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

somGetClass obtains a pointer to the receiver's class object. The **somGetClass** method is typically not overridden.

Important Note: For C and C++ programmers, SOM provides a **SOM_GetClass** macro that performs the same function. This macro should only be used **only** when absolutely necessary (that is, when a method call on the object is not possible), since it bypasses whatever semantics may be intended for the somGetClass method by the implementor of the receiver's class. Even class implementors do not know whether a special semantics for this method is inherited from ancestor classes. If you are unsure of whether the method or the macro is appropriate, you should use the method call.

Parameters

receiver	A pointer to the object whose class is desired.

Return Value

A pointer to the object's class object.

C Example

```
#include <animal.h>
main()
{
  Animal myAnimal;
  int numMethods;
  SOMClass animalClass;

  myAnimal = AnimalNew ();
  animalClass = _somGetClass (myAnimal);
  SOM_Test(animalClass == _Animal);
}
```

Original Class

SOMObject

Related Information

Macros: SOM_GetClass

somGetClassName Method

Purpose

Returns the name of the class of an object. Not generally overridden.

IDL Syntax

string somGetClassName ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetClassName** method returns a pointer to a zero-terminated string that gives the name of the class of an object.

This method is not generally overridden; it simply invokes **somGetName** on the class of the receiver. Refer to **somGetName** for more information on the returned string,

Parameters

receiver A pointer to the object whose class name is desired.

Return Value

The **somGetClassName** method returns a pointer to the name of the class.

C Example

```
#include <animal.h>
main()
{
  Animal myAnimal;
  SOMClass animalClass;
  char *className;

  myAnimal = AnimalNew();
  className = _somGetClassName(myAnimal);
  somPrintf("Class name: %s\n", className);
  _somFree(myAnimal);
}
/*
Output from this program:
Class name: Animal
*/
```

Original Class

SOMObject

Related Information

Methods: somGetName

somGetSize Method

Purpose

Returns the size of an object. Not generally overridden.

IDL Syntax

long somGetSize ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somGetSize** method returns the total amount of contiguous space used by the receiving object.

The value returned reflects only the amount of storage needed to hold the SOM representation of the object. The object might actually be using or managing additional space outside of this area.

The **somGetSize** method is not generally overridden.

Parameters

receiver	A pointer to the object whose size is desired.

Return Value

The **somGetSize** method returns the size, in bytes, of the receiver.

C Example

```
#include <animal.h>
void main()
{
  Animal myAnimal;
  long animalSize;
  myAnimal = AnimalNew();
  animalSize = _somGetSize(myAnimal);
  somPrintf("Size of animal (in bytes): %d\n", animalSize);
  _somFree(myAnimal);
}
/*
Output from this program:
Size of animal (in bytes): 8
*/
```

Original Class

SOMObject

Related Information

Methods: somGetInstancePartSize, **somGetInstanceSize**

somInit Method

Purpose

Initializes instance variables or attributes in a newly created object. Designed to be overridden. Note: The newer **somDefaultInit** method is suggested instead.

IDL Syntax

void somInit ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somInit** method is invoked to cause a newly created object to initialize its instance variables or attributes.

Note: The newer **somDefaultInit** method performs object initialization more efficiently and is now the preferred approach for overriding initialization in an implementation file. (The **somInit** method still executes correctly as before.)

Because instances of **SOMObject** do not have any instance data, the default implementation does nothing. It is provided as a convenience to class implementors so that initialization of objects can be done in a uniform way across all classes (by overriding **somInit**). This method is called automatically by **somNew** during object creation.

A companion method, **somUninit**, is called whenever an object is freed. These two methods should be designed to work together, with **somInit** priming an object for its first use, and **somUninit** preparing the object for subsequent release.

If objects of your class contain instance variables or attributes, override the **somInit** method to initialize the instance variables or attributes when instances of the class are created. When overriding this method, always call all parent (base) classes' versions of this method *before* doing your own initialization, as follows:

1. The overriding implementation should invoke the parent method for *each* parent. For users of the C or C++ implementation bindings, this can be done in either of two ways:
   (a) by calling a <*className*>**_parents_**<*methodName*> macro (which automatically invokes all parent methods) or
   (b) by calling the <*className*>**_parent_**<*parentName*>_<*methodName*> macro on each parent separately.

   For more information on parent method calls, see the topic "Extending the Implementation Template" in Chapter 5, "Implementing Classes in SOM," of the *SOM Toolkit User's Guide.*

2. The code must be written so that it can be executed multiple times without harm on the same object. This is necessary because, under multiple inheritance, parent method calls that progress up the inheritance hierarchy may encounter the same ancestor class more than once (where different inheritance paths "join" when followed backward). A check can be made to determine whether a particular invocation of **somInit** is the first on a given object by examining the contents of its instance variables; all the instance variables of a newly created SOM object are set to zero before **somInit** is invoked on that object.

More information and examples on object initialization (especially regarding the **somDefaultInit** method) are given in the topic "Initializing and Uninitializing Objects" in Chapter 5, "Implementing Classes in SOM," of the *SOM Toolkit User's Guide.*

Parameters

receiver	A pointer to the object to be initialized.

Return Value

None

C Example

Below is the implementation for a class *Animal* that introduces an attribute *sound* of type *string* and overrides **somInit** and **somUninit**, along with a main program that creates and then frees an instance of class *Animal*:

```
#define Animal_Class_Source
 #include <animal.ih>
 #include <string.h>

 SOM_Scope void SOMLINK somInit (Animal somSelf)
  {
      AnimalData *somThis = AnimalGetData (somSelf);
      Environment *ev = somGetGlobalEnvironment();
      Animal_parents_somInit (somSelf);
      if (!__get_sound(somSelf, ev)) {
         __set_sound(somSelf, ev, SOMMalloc(100));
         strcpy (__get_sound(somSelf, ev), ”Unknown Noise”);
         somPrintf (”New Animal Initialized\n”);
      }
  }

 SOM_Scope void SOMLINK somUninit (Animal somSelf)
  {
      AnimalData *somThis = AnimalGetData (somSelf);
      Environment *ev = somGetGlobalEnvironment();
      if (__get_sound(somSelf, ev)) {
        SOMFree(__get_sound(somSelf, ev);
        __set_sound(somSelf, ev, (char*)0);
        somPrintf (”Animal Uninitialized\n”);
        Animal_parents_somUninit (somSelf);
      }
  }

/* main program */
   #include <animal.h>
   void main()
   {
      Animal myAnimal;
      myAnimal = AnimalNew ();
      _somFree (myAnimal);
   }

/*
Program output:
New Animal Initialized
Animal Uninitialized
*/
```

Original Class

SOMObject

Related Information

Methods: somDefaultInit, somNew, somRenew, somDestruct, somUninit

somIsA Method

Purpose

Tests whether an object is an instance of a given class or of one of its subclasses. Not generally overridden.

IDL Syntax

boolean somIsA (in SOMClass *aClass***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

Use the **somIsA** method to determine if an object can be treated like an instance of *aClass*. SOM guarantees that if **somIsA** returns true, then the *receiver* will respond to all (static or dynamic) methods supported by *aClass*.

Parameters

receiver	A pointer to the object to be tested.
aClass	A pointer to the class that the object should be tested against.

Return Value

The **somIsA** methods returns 1 (true) if the receiving object is an instance of the specified class or (unlike **somIsInstanceOf**) of any of its descendant classes, and 0 (false) otherwise.

C Example

```
#include <dog.h>
/* --------------------------------
   Note: Dog is derived from Animal.
   -------------------------------- */
main()
{
  Animal myAnimal;
  Dog myDog;
  SOMClass animalClass;
  SOMClass dogClass;

  myAnimal = AnimalNew();
  myDog = DogNew();
  animalClass = _somGetClass (myAnimal);
  dogClass = _somGetClass (myDog);
  if (_somIsA (myDog, animalClass))
     somPrintf (”myDog IS an Animal\n”);
  else
     somPrintf (”myDog IS NOT an Animal\n”);
  if (_somIsA (myAnimal, dogClass))
     somPrintf (”myAnimal IS a Dog\n”);
  else
     somPrintf (”myAnimal IS NOT a Dog\n”);
  _somFree (myAnimal);
  _somFree (myDog);
}
/*
Output from this program:
myDog IS an Animal
myAnimal IS NOT a Dog
*/
```

Original Class

SOMObject

Related Information

Methods: somDescendedFrom, **somIsInstanceOf, somRespondsTo, somSupportsMethod**

somIsInstanceOf Method

Purpose

Determines whether an object is an instance of a specific class. Not generally overridden.

IDL Syntax

boolean somIsInstanceOf (in SOMClass *aClass***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

Use the **somIsInstanceOf** method to determine if an object is an instance of a specific class. This method tests an object for inclusion in one specific class. It is equivalent to the expression:

```
(aClass == somGetClass (receiver))
```

Parameters

receiver	A pointer to the object to be tested.
aClass	A pointer to the class that the object should be an instance of.

Return Value

The **somIsInstanceOf** method returns 1 (true) if the receiving object is an instance of the specified class, and 0 (false) otherwise.

C Example

```
#include <dog.h>
/* --------------------------------
   Note: Dog is derived from Animal.
   -------------------------------- */
main()
{
  Animal myAnimal;
  Dog myDog;
  SOMClass animalClass;
  SOMClass dogClass;

  myAnimal = AnimalNew ();
  myDog = DogNew ();
  animalClass = _somGetClass (myAnimal);
  dogClass = _somGetClass (myDog);
  if (_somIsInstanceOf (myDog, animalClass))
     somPrintf ("myDog is an instance of Animal\n");
  if (_somIsInstanceOf (myDog, dogClass))
     somPrintf ("myDog is an instance of Dog\n");
  if (_somIsInstanceOf (myAnimal, animalClass))
     somPrintf ("myAnimal is an instance of Animal\n");
  if (_somIsInstanceOf (myAnimal, dogClass))
     somPrintf ("myAnimal is an instance of Dog\n");
  _somFree (myAnimal);
  _somFree (myDog);
}
/*
Output from this program:
myDog is an instance of Dog
myAnimal is an instance of Animal
*/
```

Original Class

SOMObject

Related Information

Methods: somDescendedFrom, **somIsA**

somPrintSelf Method

Purpose

Outputs a brief description that identifies the receiving object. Designed to be overridden.

IDL Syntax

SOMObject somPrintSelf ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

somPrintSelf should output a brief string containing key information useful to identify the receiver object, rather than a complete dump of the receiver object state as provided by **somDumpSelfInt**. The **somPrintSelf** method should use the character output routine **SOMOutCharRoutine** (or any of the **somPrintf** functions) for this purpose. The default implementation outputs the name of the receiver object's class and the receiver's address in memory.

Because the most specific identifying information for an object will often be found within instance data introduced by the class of an object, it is likely that a class implementor that overrides this method will not need to invoke parent methods in order to provide a useful string identifying the receiver object.

Parameters

receiver	A pointer to the object to be described.

Return Value

The **somPrintSelf** method returns a pointer to the receiver object as its result.

C Example

```
#include <animal.h>
main()
{
  Animal myAnimal;
  myAnimal = AnimalNew ();
  /* ... */
  _somPrintSelf (myAnimal);
  _somFree (myAnimal);
}
/*
Output from this program:

{An instance of class Animal at address 0001CEC0}
*/
```

Original Class

SOMObject

Related Information

Methods: somDumpSelf, **somDumpSelfInt**

somResetObj Method

Purpose

Resets an object's class to its true class after use of the **somCastObj** method.

IDL Syntax

boolean somResetObj ( );

Description

The **somResetObj** method resets an object's class to its true class after use of the **somCastObj** method.

Parameters

receiver A pointer to a SOM object.

Return Value

The **somResetObj** method returns 1 (TRUE) always.

Example

```
#include <som.h>
main()
{
   SOMClassMgr cm = somEnvironmentNew();
   SOM_Test(1 == _somCastObj(cm, _SOMObject));
   _somDumpSelf(cm, 0));
   SOM_Test(1 == _somResetObj(cm));
   _somDumpSelf(cm, 0);
}

/* output:
 *  {An instance of class SOMClassMgr->SOMObject
 *    at address 20061268
 *  }
 *  {An instance of class SOMClassMgr at address 20061268
 *    ... <SOMClassMgr State Information> ...
 *  }
 */
```

Original Class

SOMObject

Related Information

Methods: somCastObj

somRespondsTo Method

Purpose

Tests whether the receiving object supports a given method. Not generally overridden.

IDL Syntax

boolean somRespondsTo (in somId *methodId***);**

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somRespondsTo** method tests whether a specific (static or dynamic) method can be invoked on the receiver object. This test is equivalent to determining whether the class of the receiver *supports* the specified method on its instances.

Parameters

receiver A pointer to the object to be tested.

methodId A **somId** that represents the name of the desired method.

Return Value

The **somRespondsTo** method returns TRUE if the specified method can be invoked on the receiving object, and FALSE otherwise.

C Example

```
/* -----------------------------------------------
   Note: Animal supports a setSound method;
         Animal does not support a doTrick method.
   ----------------------------------------------- */
#include <animal.h>
main()
{
  Animal myAnimal;
  char *methodName1 = ”setSound”;
  char *methodName2 = ”doTrick”;

  myAnimal = AnimalNew();
  if (_somRespondsTo(myAnimal, SOM_IdFromString(methodName1)))
     somPrintf(”myAnimal responds to %s\n”, methodName1);
  if (_somRespondsTo(myAnimal, SOM_IdFromString(methodName2)))
     somPrintf(”myAnimal responds to %s\n”, methodName2);
  _somFree(myAnimal);
}
/*
Output from this program:
myAnimal responds to setSound
*/
```

Original Class

SOMObject

Related Information

Methods: somSupportsMethod

somUninit Method

Purpose

Un-initializes the receiving object. Designed to be overridden by class implementors. Not normally invoked directly by object clients.

IDL Syntax

void somUninit ( );

Note: For backward compatibility, this method does *not* take an **Environment** parameter.

Description

The **somUninit** method performs the inverse of object initialization. Class implementors that introduce instance data that points to allocated storage should override **somUninit** so allocated storage can be freed when an object is freed.

This method is called automatically by **somFree** to clean up anything necessary (such as extra storage dynamically allocated to the object) before **somFree** releases the storage allocated to the object itself.

Code responsible for freeing an object must first know that there will be no further references to this object. Once this is known, this code would normally invoke **somFree** (which calls **somUninit**). In cases where **somRenew** was used to create an object instance, however, **somFree** cannot be called (for example, the storage containing the object may simply be a location on the stack), and in this case, **somUninit** must be called explicitly.

When overriding this method, always call the parent-class versions of this method *after* doing your own un-initialization. Furthermore, just as with **somInit**, because your method may be called multiple times (due to multiple inheritance), you should zero out references to memory that is freed, and check for zeros before freeing memory and calling the parent methods.

Parameters

receiver A pointer to the object to be un-initialized.

Return Value

None

C Example

Following is the implementation for a class *Animal* that introduces an attribute *sound* of type *string* and overrides **somInit** and **somUninit**, along with a main program that creates and then frees an instance of class *Animal*:

```
#define Animal_Class_Source
 #include <animal.ih>
 #include <string.h>

 SOM_Scope void SOMLINK somInit (Animal somSelf)
  {
      AnimalData *somThis = AnimalGetData (somSelf);
      Environment *ev = somGetGlobalEnvironment();
      Animal_parents_somInit (somSelf);
      if (!__get_sound(somSelf, ev)) {
         __set_sound(somSelf, ev, SOMMalloc(100));
         strcpy (__get_sound(somSelf, ev), "Unknown Noise");
         somPrintf ("New Animal Initialized\n");
      }
  }

 SOM_Scope void SOMLINK somUninit (Animal somSelf)
  {
      AnimalData *somThis = AnimalGetData (somSelf);
      Environment *ev = somGetGlobalEnvironment();
      if (__get_sound(somSelf, ev)) {
        SOMFree(__get_sound(somSelf, ev);
        __set_sound(somSelf, ev, (char*)0);
        somPrintf ("Animal Uninitialized\n");
        Animal_parents_somUninit (somSelf);
      }
  }

/* main program */
   #include <animal.h>
   void main()
   {
      Animal myAnimal;
      myAnimal = AnimalNew ();
      _somFree (myAnimal);
   }

/*
Program output:
New Animal Initialized
Animal Uninitialized
*/
```

Original Class

SOMObject

Related Information

Methods: somInit, **somNew**, **somRenew**