SOMobjects Developer Toolkit

Quick Reference Guide

Syntax reference for the classes, methods, functions, macros and SOM Compiler of the System Object Model and its accompanying frameworks

Version 2.1
October 1994

Version 2.1 (October 1994)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THE PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate AIX, OS/2, or Windows programming techniques. You may copy and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the AIX, OS/2, or Windows application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "©(your company name) (year) All Rights Reserved."

However, the following copyright notice protects this documentation under the Copyright laws of the United States and other countries which prohibit such actions as, but not limited to, copying, distributing, modifying, and making derivative works.

© Copyright International Business Machines Corporation, 1991 — 1994. All rights reserved.

The term "IBM" is a registered trademark and "SOMobjects" and "System Object Model" are trademarks of International Business Machines Corporation.

Notice to US Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

SOMobjects Developer Toolkit Quick Reference Guide

Contents

SOM Compiler Quick Reference

SOM Compiler 'sc' or 'somc' command

Usage:

sc [–C:D:E:I:S:U:V:c:d:h:i:m:p:r:s:u:v:w] f1 f2 ... (On AIX or OS/2)
somc [–C:D:E:I:S:U:V:c:d:h:i:m:p:r:s:u:v:w] f1 f2 ... (On Windows)

Compiles the specified .idl file(s) and produces binding files as the options, –m modifiers, and environment variables direct.

where:

–C <n>	Sets size of comment buffer (default: 32767).
–D <DEFINE>	Has same effect as –D option for cpp.
–E <var>=<value>	Sets an environment variable.
–I <INCLUDE>	Has same effect as –I option for cpp.
–S <n>	Sets size of string buffer (default: 32767)
–U <UNDEFINE>	Has same effect as –U option for cpp.
–V	Show version number of compiler.
–c	Ignore all comments.
–d <dir>	Send output to directory for each emitted file.
–h	Displays a listing of this option list.
–i <file>	Use filename as specified (not a default .idl file).
–m <name[=value]>	Adds a global modifier (see valid Modifiers, below).
–p	Denotes shorthand for the option –D__PRIVATE__.
–r	Check release order entries exist (default: FALSE).
–s <string>	Replaces SMEMIT variable with <string>. Note: Windows users should *not* use quotes in the string; for example, `somc –sh;ih *.idl` Correct `somc –s"h;ih" *.idl` Incorrect
–u	Updates Interface Repository.
–v	Sets verbose debugging mode (default: FALSE).
–w	Don't display warnings (default: FALSE).

Modifiers for –m option

addprefixes	Adds 'functionprefix' to method names in the template file.
addstar	Adds '*****' to C bindings for interface references.
corba	Checks the source for CORBA compliance.

csc	Forces running of the OIDL compiler.
emitappend	Appends the emitted files at the end of the existing file.
noheader	Don't add a header to the emitted file.
noint	Don't warn about "int" causing portability problems.
nolock	Don't lock the IR during update.
nopp	Don't run the source through the pre-processor.
notc	Don't use typecodes for emit information.
nouseshort	Don't generate short names for types
pp=<path>	Specifies a local pre-processor to use.
tcconsts	Generate CORBA TypeCode constants.

Note: All command-line modifiers can be set in the environment by changing them to UPPERCASE and prepending "SM" to them.

Environment variables

SMEMIT=[h; ih; c; xh; xih; xc; def; ir; pdl]	Determines which emitters to run (default: h; ih).
SMINCLUDE=<dir1>[;<dir2>]+	Determines where to search for .idl and .efw files.
SMKNOWNEXTS=ext[;ext]+	Adds headers to user-written emitters.
SMTMP=<dir>	Sets a directory to hold intermediate files.
SOMIR=<path>[;<path>]+	Gives a list of Interface Repositories to search.

Pragmas

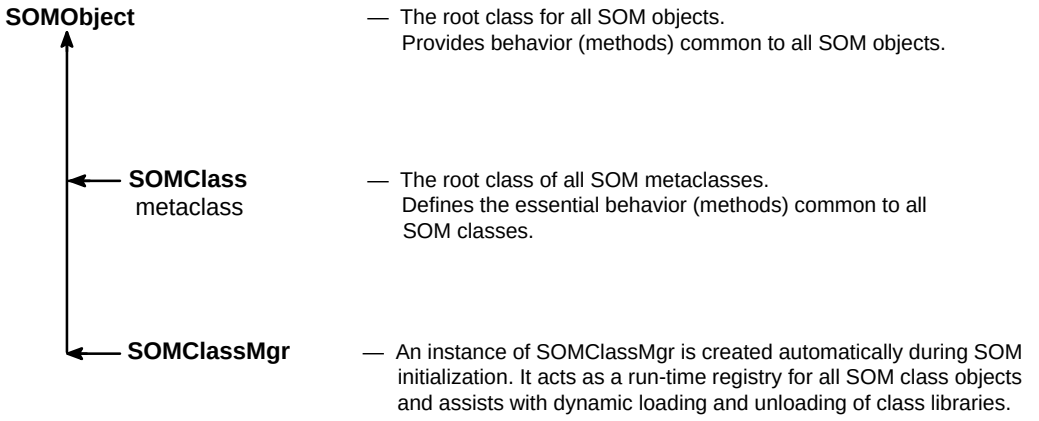
#pragma somemittypes on	Turns on emission of global types.
#pragma somemittypes off	Turns off emission of global types.
#pragma modifier <modifier stm>;	Use this statement instead of the definition for the same 'modifier' in the implementation section of the .idl file.

Other commands

pdl –d<dir> ***.idl**	Install public versions of the .idl files in the indicated directory.
ctoi *.csc	Convert file from .csc to .idl. (See Appendix B of the *SOMobjects Developer Toolkit Users Guide*.)

SOM Kernel Quick Reference

Class Organization



SOM Functions

(see somapi.h and somcorba.h)
Note: function prototypes are given using C syntax, assuming the type environment created by: #include <som.h>. Pointers to objects that support an interface are typed using the interface name, without an explicit asterisk (*).

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

Invokes (on the object pointed to by objPtr) the apply stub for the method described by the structure pointed to by the somMethodDataPtr. The result of the method invocation is stored in the memory location pointed to by retVal (which cannot be null). The va_list must include objPtr as its first entry.

void **somBeginPersistentIds** ();

Tells SOM to begin a “persistent ID interval.”

void **somBuildClass** (
unsigned long inheritVars,
somStaticClassInfoPtr sciPtr,
long majorVersion,
long minorVersion);

Automates the process of building a new SOM class object; used by C and C++ implementation bindings.

somId **somCheckId** (somId id);

Registers a SOM ID.

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

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

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

Determines whether two SOM IDs represent the same string. Returns true (1) if they are the same and false (0) otherwise.

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

Supports access to instance data within an object. Used by C and C++ implementation bindings.

void **somEndPersistentIds** ();	Tells SOM to end a “persistent ID interval.”
void **somEnvironmentEnd** ( );	Provides general cleanup for applications.
SOMClassMgr **somEnvironmentNew** ();	Initializes the SOM runtime environment.
void **somExceptionFree** (Environment *ev);	Frees the memory held by the Exception structure within an Environment structure.
string **somExceptionId** (Environment *ev);	Gets the name of the exception contained in an Environment structure.
somToken **somExceptionValue** (Environment *ev);	Gets the value of the exception contained in an Environment structure.
Environment ***somGetGlobalEnvironment** ();	Returns a pointer to the current global Environment structure.
somId **somIdFromString** (string aString);	Returns the SOM ID corresponding to a given text string.
boolean **somIsObj** (somToken memPtr);	Failsafe routine to determine whether a pointer references a valid SOM object.
long **somLPrintf** (long level, string fmt, ...);	Prints a formatted string in the manner of the C printf function, at the specified indentation level.
SOMClassMgr ***somMainProgram** ( );	Performs SOM initialization on behalf of a new program.
somMethodPtr **somParentNumResolve** (somMethodTabs parentMtabs, int parentNum, somMToken mToken);	Obtains a pointer to a procedure that implements the static method identified by mToken. Used by C and C++ implementation bindings to support parent method calls for multiple-inheritance classes.
somMethodPtr **somParentResolve** (somMethodTabs parentMtab, somMToken mToken);	Obtains a pointer to the procedure that implements the static method identified by mToken. Used by old binaries to support parent method calls for single inheritance classes.
void **somPrefixLevel** (long level);	Outputs blanks to prefix a line at the indicated level.
long **somPrintf** (string fmt, ...);	Prints a formatted string in the manner of the C printf function.
int **somRegisterId** (somId id);	Registers a SOM ID and determines whether or not it was previously registered.
somMethodPtr **somResolve** (SOMObject objPtr, somMToken mToken);	Obtains a pointer to the procedure that implements the static method identified by mToken for a particular SOM object.
somMethodPtr **somResolveByName** (SOMObject objref, string methodName);	Obtains a pointer to the procedure that implements a (static or dynamic) method for a particular SOM object.
void **somSetException** (Environment *ev, enum exception_type major, string exceptionName, somToken params);	Sets an exception value in an Environment structure.

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

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

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

Changes the behavior of the somPrintf function.

string **somStringFromId** (somId id);

Returns the string that a SOM ID represents.

unsigned long **somTotalRegIds** ();

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

unsigned long **somUniqueKey** (somID id);

Returns the unique key associated with a SOM ID.

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

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

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

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

string **(*SOMClassInitFuncName)** ( );

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

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

Unloads a dynamically linked library (DLL). Replaceable.

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

Handles an error condition. Replaceable.

void **(*SOMFree)** (somToken memPtr);

Frees the specified block of memory. Replaceable.

SOMEXTERN void SOMLINK **SOMInitModule** (
long majorVersion,
long minorVersion,
string className);

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

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

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

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

Allocates the specified amount of memory. Replaceable.

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

Prints a character. Replaceable.

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

Changes the size of a previously allocated region of memory. Replaceable.

C/C++ Macros

(see somcdev.h)
Note: Macro arguments are given using C argument declaration syntax for arguments that are C expressions (for example,, “string className” denotes an argument expression that evaluates to a char* value in C), whereas macro arguments that are simply tokens manipulated by the preprocessor in order to create C expressions are identified with the prefix <token>.

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

Asserts that a boolean condition is true — that is, a C/C++ expression evaluates to a non-zero value.

void **SOM_ClassLibrary** (
string “ libname.dll ”);

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

Environment ***
SOM_CreateLocalEnvironment ();

Creates and initializes a local Environment structure that can be passed to methods as the Environment argument.

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

Destroys a local Environment structure; calls somExceptionFree and SOMFree.

void **SOM_Error** (
long errorCode);

Reports an error condition.

void **SOM_Expect** (
boolean expression);

Asserts that a boolean condition is expected to be true.

SOMClass **SOM_GetClass** (
SOMObject objPtr);

Returns a pointer to the class object of which a SOM object is an instance. Invokes no methods.

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

Initializes a locally declared Environment structure that can be passed to methods as the Environment argument.

SOMClassMgr SOM_MainProgram ( );

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

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

Used to turn off method debugging. The macro arguments are not C/C++ expressions, but simply text used by the preprocessor to create expressions.

somMethodPtr **SOM_ParentNumResolve** (
<token> className,
long parentNum,
somMethodTabs mtabs,
<token> methodName);

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.

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

Obtains a pointer to a static method procedure.

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

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

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

Provides a convenience macro for invoking :the somSubstituteClass method.

void **SOM_Test** (boolean expression);	Tests whether a boolean condition is true; if not, a fatal error is raised.
void **SOM_TestC** (boolean expression);	Tests whether a boolean condition is true; if not, a warning message is output.
void **SOM_UninitEnvironment** (Environment * ev);	Uninitializes a local Environment structure; calls somExceptionFree (but *not* SOMFree).
void **SOM_WarnMsg** (string msg);	Reports a warning message.

SOMClass class

(see somcls.idl)
Note: All following method prototypes for the SOM kernel classes are expressed using IDL syntax for OIDL callstyle methods. That is, the receiver of the method is implicit, and the two possible CORBA implicit arguments (environment and context) are not passed. For example, to call the _get_somInstanceDataOffsets method from C (assuming an #include <somcls.h>) the following prototype specifies that a programmer could write the statement: __get_somInstanceDataOffsets(classPtr).

somOffsets **_get_somInstanceDataOffsets**();	Returns a sequence of structures, each of which indicates a class and the offset to the beginning of the instance data introduced by the indicated class in an instance of the receiver class.
void **somAddDynamicMethod** (in somId methodId, in somId methodDescriptor, in somMethodPtr method, in somMethodPtr applyStub);	Adds a new dynamic instance method to a class. Dynamic methods must be invoked using name-lookup or dispatch resolution.
string **somAllocate** (in long size);	Supports class-specific memory allocation for class instances. Cannot be overridden.
boolean **somCheckVersion** (In long majorVersion, In long minorVersion);	Checks a class for compatibility with the specified major and minor version numbers.
void **somClassReady** ();	Indicates that a class has been constructed and is ready for normal use.
void **somDeallocate** (in string memPtr);	Frees memory originally allocated by the somAllocate method from the same class object. Cannot be overridden.
boolean **somDescendedFrom** (in SOMClass clsPtr);	Tests whether a specified class is derived from the receiving class.
boolean **somFindMethod** (in somId methodId, out somMethodPtr m);	Finds the method procedure that implements the indicated method for a class of objects, and indicates whether the method is a static.
boolean **somFindMethodOk** (in somId methodId, out somMethodPtr m);	As above; but if the method is not supported, this method raises an error and halts execution.
somMethodPtr **somFindSMethod** (in somId methodId);	Finds the method procedure for a static method.

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

Like somFindSmethod; but if the desired static method is not supported, an error is raised and execution is halted.

long **somGetInstancePartSize** ();

Returns the size of the instance variables introduced by a class, in all instances of that class.

long **somGetInstanceSize** ();

Returns the size of an instance of a class.

somDToken **somGetInstanceToken** ();

Returns a token for the instance data introduced by a class.

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

Returns an access token for an instance variable.

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

Gets the method data for the indicated method, which must have been introduced by the receiver or an ancestor of the receiver.

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

Returns the method descriptor for a method.

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

Returns the index for the specified method. Can be used as input to somGetNthMethodData.

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

Returns a static method's access token.

string **somGetName** ();

Obtains the name of the receiving class.

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

Loads the somMethodData structure for the nth (static or dynamic) method known to the receiver class, given the index to the method. See also somGetNumMethods and somGetNumStaticMethods.

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

Returns the method ID of the nth (static or dynamic) method known to this class. Also loads the location pointed to by the passed descriptor address with a somId for the method descriptor.

long **somGetNumMethods** ();

Obtains the number of methods available for a class.

long **somGetNumStaticMethods** ();

Obtains the number of static methods available for a class.

SOMClassSequence **somGetParents** ();

Returns a sequence containing pointers to the parents of the receiver class.

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

Gets the major and minor version numbers of the receiving class's implementation code.

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

Like somFindSMethod except dynamic methods can also be returned. Used by bindings for name-lookup resolution.

SOMObject **somNew** ();

Creates a new instance of the receiving class and calls somInit to initialize it.

SOMObject **somNewNoInit** ();

Creates a new instance of a class, but does not initialize it.

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

Creates a new object instance of the receiver class in the indicated memory location. Zeros memory, and calls somInit to re-initialize the object.

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

Like somRenew, but does *not* call somInit to initialize it.

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

Like somRenew, but does not zero memory.

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

Like somRenew, but does not call somInit and does not zero memory.

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

Indicates whether instances of a given class support a given (static or dynamic) method.

SOMClassMgr class

(see somcm.idl)
Note: Method prototypes for the SOM kernel classes are expressed using IDL syntax for OIDL callstyle methods. That is, the receiver of the method is implicit, and the two possible CORBA implicit arguments (environment and context) are not passed.

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

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

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

Finds a class object, given its ID. If necessary, loads the class and initializes it.

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

Finds a class object for a class when the correct file is known beforehand. If necessary, loads the class and initializes it.

string **somGetInitFunction** ();

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

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

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

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

Dynamically loads a class.

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

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

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

Transfers SOM class registry to another SOMClassMgr instance.

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

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

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

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

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

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

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

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

SOMObject class

(see somobj.idl)
Note: Method prototypes for the SOM kernel classes are expressed using IDL syntax for OIDL callstyle methods. That is, the receiver of the method is implicit, and the two possible CORBA implicit arguments (environment and context) are not passed.

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

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

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

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

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

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

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

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

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

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.

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

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.

void **somDestruct** (
in octet dofree,
inout somDestructCtrl ctrl);

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. It is recommended that somDestruct always be overridden. Not normally invoked directly by object clients.

boolean **somDispatch** (
out somToken retValue,
in somId methodId,
in va_list args);

Invokes the indicated method on the receiver. For static methods, resolution of the indicated method is performed using the method table of the receiver's class. The result returned by the method is stored into the location pointed to by retValue. The va_list must include the target object as well as the arguments.

boolean **somClassDispatch** (
in SOMClass clsObj,
out somToken retValue,
in somId methodId,
in va_list args);

Like somDispatch, except that static method resolution is performed using the instance method table of the specified class (clsObj).

somToken **somDispatchA** (
in somId methodId,
in somId descriptor,
in va_list args);

Obsolete. Like somDispatch, but restricted to methods that return a pointer. The va_list must not include the target object. This method has been superseded by somDispatch and is included solely for backward compatibility.

double **somDispatchD** (
in somId methodId,
in somId descriptor,
in va_list args);

Obsolete. As above, but restricted to methods that return a double precision floating point.

long **somDispatchL** (
in somId methodId,
in somId descriptor,
in va_list args);

Obsolete. As above, but restricted to methods that return a long.

void **somDispatchV** (
in somId methodId,
in somId descriptor,
in va_list args);

Obsolete. As above, but restricted to methods that return void.

void **somDumpSelf** (
in long level);

Clients of an object use this method to write out a detailed description of the object.

void **somDumpSelfInt** (
in long level);

Implementors of a class override this method to support somDumpSelf.

void **somFree** ();

Releases the storage used by an object and frees the object.

SOMClass **somGetClass** ();

Returns a pointer to an object's class object.

string **somGetClassName** ();

Returns the name of the class of an object.

long **somGetSize** ();

Returns the size of an object.

void **somInit** ();

Implementors of a class may override this method to initialize instance variables or attributes introduced by the class.

boolean **somIsA** (
in SOMClass aClass);

Tests whether the receiving object is an instance of the indicated class or of one of its subclasses.

boolean **somIsInstanceOf**(
in SOMClass aClass);

Tests whether the receiving object is an instance of the indicated class.

SOMObject **somPrintSelf** ();

Implementors of a class may override this method to write a brief description of the receiving object. A pointer to the receiving object is returned as the result of the method call.

boolean **somResetObj** ( );

Resets an object's class to its true class after use of the somCastObj method.

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

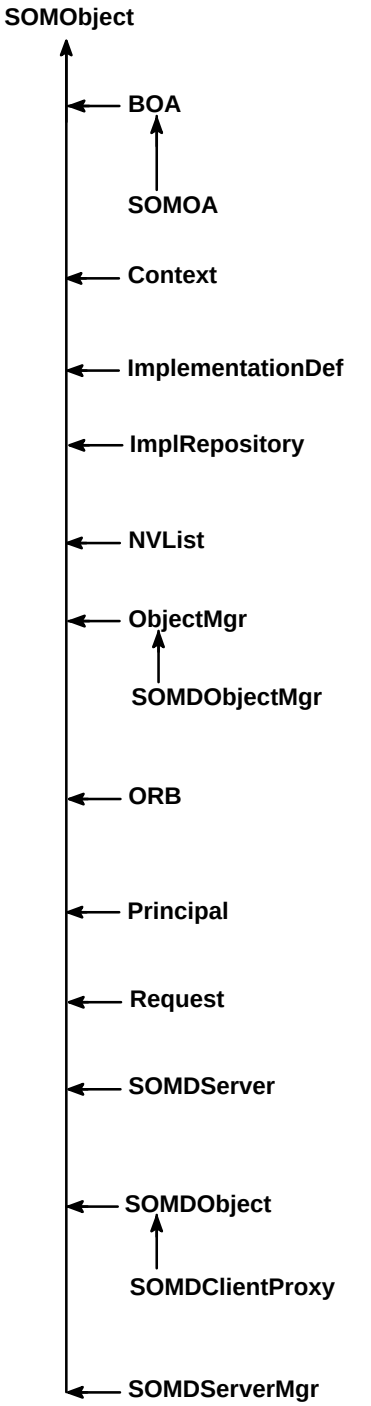
Tests whether an object can respond to an invocation of the indicated method.

void **somUninit** ();

Uninitializes the receiving object. Implementors of a class may override this method to free any allocated memory pointed to by instance data or attributes introduced by the class.

DSOM Framework Quick Reference

Class Organization



– *Basic Object Adapter.* Abstract class which defines basic interfaces that a server process uses to access services of an Object Request Broker like DSOM. Defines methods for creating and exporting object references, registering and activating implementations, and authenticating requests.

– *SOM Object Adapter*. Subclass of BOA. Implements BOA methods, and introduces methods for receiving and dispatching requests.

– *Context.* List of properties (Environment variables) that can be passed in a method invocation. Each property consists of a name and a string value.

– *Implementation Definition*. Attributes define the implementation of a server.

– *Implementation Repository.* Defines and implements methods used to query and update the DSOM implementation repository, which stores copies of the ImplementationDef objects.

– *Named Value List.* Implements a list of <name,value> pairs. NVLists are used in building Request objects and Context objects.

– *Object Manager.* Abstract class which defines basic interfaces for creating objects, destroying objects, and mapping between objects and their ids.

– *DSOM Object Manager*. Provides a DSOM-specific implementations of the ObjectMgr abstract class. Also, introduces methods for finding DSOM server implementations, by id, alias, and class.

– *Object Request Broker.* Defines and implements various utility methods as defined in the CORBA 1.1 specification. Included are methods for converting object references to and from strings, for creating NVLists to be used in building specific Requests, and for obtaining a default Context object.

– *Principal.* Attributes contain the ids of the user and host from which a request originated. Used by application for access control checking.

– *Request*. Represents a method call and parameters, in the Dynamic Invocation Interface. Includes methods used to invoke the request, either synchronously (waits for a response) or asynchronously.

– *DSOM Server*. Base class which defines and implements methods for managing objects in a server. Includes methods for object creation and deletion, and mapping between SOM objects and object references. Also supports intercepting and redispatching method calls.

– *DSOM Object.* Implements an "object reference" in DSOM. Includes methods for getting an implementation or interface from a reference, testing whether it is nil, and duplicating or releasing the reference.

– *DSOM Client Proxy*. Subclass of SOMDObject. This "mixin" class is used to implement proxy objects in client processes. Provides the remote dispatching function used to forward requests to target objects. Also supports the construction of dynamic requests to be invoked against the proxy object.

– *DSOM Server Mgr*. Provides a programmatic interface to manage server processes. Server processes that can be managed are limited to those in the Implementation Repository (as set by the SOMDDIR environment variable).

DSOM functions

(see request.h)

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

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

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

Initiates multiple Requests in parallel.

(see somdext.h)

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

Frees memory allocated by DSOM for return values and output arguments.

void **somdExceptionFree** (
Environment *ev);

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

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

Initializes DSOM in the calling process.

void **SOMD_NoORBfree** ();

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

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

Registers a callback function for handling DSOM request events.

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

Frees system resources allocated for use by DSOM.

DSOM macros

(see cntxt.h)

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

Deletes a Context object.

(see request.h)

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

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

BOA class

(see boa.idl)

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

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

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

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

void **deactivate_impl** (in ImplementationDef impl);	Indicates that a server implementation is no longer ready to process requests.
void **deactivate_obj** (in SOMDObject obj);	Indicates that an object server is no longer ready to process requests. *(Not implemented.)*
void **dispose** (in SOMDObject obj);	Destroys an object reference.
ReferenceData **get_id** (in SOMDObject obj);	Returns reference data associated with the referenced object.
Principal **get_principal** (in SOMDObject obj, in Environment* req_ev);	Returns the ID of the principal that issued the request.
void **impl_is_ready** (in ImplementationDef impl);	Indicates that the server implementation is ready to process requests.
void **obj_is_ready** (in SOMDObject obj, in ImplementationDef impl);	Indicates that the object (server) is ready to process requests. *(Not implemented.)*
void **set_exception** (in exception_type major, in string except_name, in void *param);	Returns an exception to a client.

Context class

(see cntxt.idl)

ORBStatus **create_child** (in Identifier ctx_name, out Context child_ctx);	Creates a child of a Context object.
ORBStatus **delete_values** (in Identifier prop_name);	Deletes property value(s).
ORBStatus **destroy** (in Flags del_flag);	Deletes a Context object.
ORBStatus **get_values** (in Identifier start_scope, in Flags op_flags, in Identifier prop_name, out NVList values);	Retrieves the specified property values.
ORBStatus **set_one_value** (in Identifier prop_name, in string value);	Adds a single property to the specified Context object.
ORBStatus **set_values** (in NVList values);	Adds/changes one or more property values in the specified Context object.

ImplementationDef class

(see impldef.idl)

attribute string **impl_id**;	Contains the DSOM-generated identifier for a server implementation.
attribute string **impl_alias**;	Contains the “alias” (user-friendly name) for a server implementation.

attribute string **impl_program**;
Contains the full pathname of the program which will be executed by the process for this server.

attribute Flags **impl_flags**;
Contains a bit-vector of flags used to identify server options (for example, multi-threading).

attribute string **impl_server_class**;
Contains the name of the SOMDServer class or subclass created by the server process.

attribute string **impl_refdata_file**;
Contains the full pathname of the file used to store ReferenceData for the server.

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

attribute string **impl_hostname**;
Contains the hostname of the machine where the server is located.

ImplRepository class

(see implrep.idl)

void **add_class_to_impldef** (
in ImplId implid,
in string classname);
Associates a class with a server.

void **add_impldef** (
in ImplementationDef impldef);
Adds an implementation definition to the Implementation Repository.

void **delete_impldef** (
in ImplId implid);
Deletes an implementation definition from the Implementation Repository.

ORBStatus **find_all_impldefs** (
out sequence<ImplementationDef>
outimpldefs);
Returns all the implementation definitions in the Implementation Repository.

sequence<string>
find_classes_by_impldef (
in ImplId implid);
Returns a sequence of class names associated with a server.

ImplementationDef **find_impldef** (
in ImplId implid);
Returns a server implementation definition given its ID.

ImplementationDef **find_impldef_by_alias** (
in string alias_name);
Returns a server implementation definition given its user-friendly alias.

sequence<ImplementationDef>
find_impldef_by_class (
in string classname);
Returns a sequence of implementation definitions for servers that are associated with a specified class.

void **remove_class_from_all** (
in string className);
Removes the association of a particular class from all servers.

void **remove_class_from_impldef** (
in ImplId implid,
in string classname);
Removes the association of a particular class with a server.

void **update_impldef** (
in ImplementationDef impldef);
Updates an implementation definition in the Implementation Repository.

NVList class

(see nvlist.idl)

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

Adds an item to the specified NVList.

ORBStatus **free** ( );

Frees a specified list structure.

ORBStatus **free_memory** ( );

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

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

Returns the total number of items allocated for a list.

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

Returns the contents of a specified list item.

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

Sets the contents of an item in an NVList.

ObjectMgr class

(see om.idl)

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

Requests destruction of the target object.

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

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

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

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

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

Returns a new object of the named type and implementation.

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

Indicates that the client has finished using the object.

ORB class

(see orb.idl)

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

Creates an NVList of the specified size.

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

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

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

Returns the default process Context object.

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

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

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

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

Principal class

(see principl.idl)

attribute string **userName**;

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

attribute string **hostName**;

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

Request class

(see request.idl)

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

Incrementally adds an argument to a Request object.

ORBStatus **destroy** ( );

Deletes the memory allocated by DSOM for a Request object.

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

Determines whether an asynchronous Request has completed.

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

Invokes a Request synchronously, waiting for the response.

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

Invokes a Request asynchronously.

SOMDClientProxy class

(see somdcprx.idl)

void **somdProxyFree** ( );

Executes somFree on the local proxy object.

SOMClass **somdProxyGetClass** ( );

Returns the class object for the local proxy object.

string **somdProxyGetClassName** ( );

Returns the class name for the local proxy object.

void **somdReleaseResources** ();

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

void **somdTargetFree** ( );

Forwards the somFree method call to the remote target object.

SOMClass **somdTargetGetClass** ( );

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

string **somdTargetGetClassName** ( );

Returns the class name for the remote target object.

SOMDObject class

(see somdobj.idl)

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

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

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

Creates an argument list appropriate for the specified operation (method).

SOMDObject **duplicate** ( );

Makes a duplicate of an object reference.

ImplementationDef **get_implementation** ( );

Returns the implementation definition for the referenced object.

InterfaceDef **get_interface** ( );

Returns the interface definition object for the referenced object.

boolean **is_constant** ( );

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

boolean **is_nil** ( );

Tests to see if the object reference is nil.

boolean **is_SOM_ref** ( );

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

boolean **is_proxy** ( );

Tests to see if the object reference is a proxy.

void **release** ( );

Releases the memory associated with the specified object reference.

SOMDObjectMgr class

(see somdom.idl)

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

Finds a server capable of creating the specified object.

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

Finds a server given its ImplementationDef name (alias).

sequence<SOMDServer>
somdFindServersByClass (
in Identifier objclass);

Finds all servers capable of creating a particular object.

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

Finds a server, given its ImplementationDef ID.

attribute boolean **somd21somFree**

Determines whether somFree, when invoked on a proxy object, will free the proxy object along with the remote object.

SOMDServer class

(see somdserv.idl)

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

Creates an object of the specified class.

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

Deletes the specified object.

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

Dispatch a method on the specified SOM object.

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

Creates a class object for the specified class.

boolean **somdObjReferencesCached ( );**

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

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

Returns an object reference corresponding to the specified SOM object.

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

Returns the SOM object corresponding to the specified object reference.

SOMDServerMgr class

(see servmgr.idl)

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

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

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

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

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

Determines whether a server process is enabled or not.

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

Queries the state of a server process.

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

Restarts a server process.

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

Stops a server process.

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

Starts a server process.

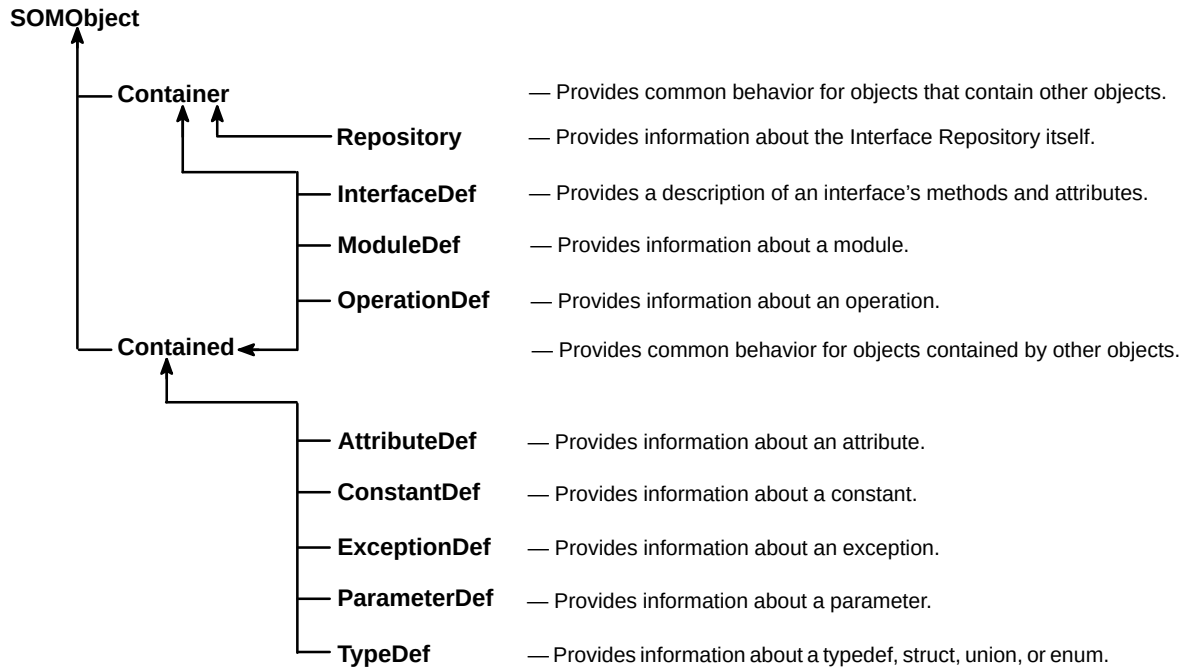
SOMOA class

(see somoa.idl)

void **activate_impl_failed** (in ImplementationDef implDef, in long rc);	Sends a message to the DSOM daemon indicating that the server process did not activate.
void **change_id** (in SOMDObject objref, in ReferenceData id);	Changes the reference data associated with an object.
SOMDObject **create_constant** (in ReferenceData id, in InterfaceDef intf, in ImplementationDef impl);	Creates a “constant” object reference.
SOMDObject **create_SOM_ref** (in SOMObject somobj, in ImplementationDef impl);	Creates a simple, transient DSOM reference to a SOM object.
ORBStatus **execute_next_request** (in Flags waitFlag);	Receive a request message, execute the request, and return the result to the caller.
ORBStatus **execute_request_loop** (in Flags waitFlag);	Continuously receives request messages, executing them and returning results.
SOMObject **get_SOM_object** (in SOMDObject somref);	Get the SOM object associated with a simple DSOM reference.

Interface Repository Framework Quick Reference

Class Organization



Contained class

(see containd.idl)

Description **describe** ( );

Returns a structure containing all of the information defined in the IDL specification that corresponds to a specified Contained object in the Interface Repository.

sequence(Container) **within** ( );

Returns a list of objects (in the Interface Repository) that contain a specified Contained object.

Container class

(see containr.idl)

sequence(Contained) **contents** (
in InterfaceName limit_type,
in boolean exclude_inherited);

Returns a sequence indicating the objects contained within a specified Container object of the Interface Repository.

sequence(ContainerDescription)
describe_contents (
in InterfaceName limit_type,
in boolean exclude_inherited,
in long max_returned_objs);

Returns a sequence of descriptions of the objects contained within a specified Container object of the Interface Repository.

```
sequence(Contained)  lookup_name (
        in Identifier search_name,
        in long levels_to_search,
        in InterfaceName limit_type,
        in boolean exclude_inherited);
```

Locates an object by name within a specified Container object of the Interface Repository, or within objects contained in the Container object.

InterfaceDef class

(see intfacdf.idl)

```
FullInterfaceDescription
        describe_interface ( );
```

Returns (from the Interface Repository) a description of all the methods and attributes of an interface definition.

Repository class

(see repostry.idl)

```
Contained  lookup_id (
        in RepositoryId search_id);
```

Returns the object having a specified RepositoryId.

```
string  lookup_modifier (
        in RepositoryId id,
        in string modifier);
```

Returns the value of a given SOM modifier for a specified object [that is, for an object that is a component of an IDL interface (class) definition maintained within the Interface Repository].

```
void  release_cache ( );
```

Releases implicitly referenced objects in the internal Interface Repository cache.

Functions

(see somtc.h)

```
short  TypeCode_alignment ( );
```

Returns the alignment value from a given TypeCode.

```
TypeCode  TypeCode_copy ( );
```

Creates a new copy of a given TypeCode.

```
boolean  TypeCode_equal (
        TypeCode tc2);
```

Compares two TypeCodes for equality.

```
void  TypeCode_free ( );
```

Destroys a given TypeCode by freeing all of the memory used to represent it.

```
TCKind  TypeCode_kind ( );
```

Categorizes the abstract data type described by a TypeCode.

```
enum  TCKind {
        tk_null,  tk_void,
        tk_short,  tk_long,
        tk_ushort, tk_ulong,
        tk_float,  tk_double,
        tk_boolean,  tk_char,
        tk_octet,  tk_any,
        tk_TypeCode,  tk_Principal,
        tk_objref,  tk_struct,
        tk_union,  tk_enum,
        tk_string, tk_sequence,
        tk_array,  tk_pointer,
        tk_self, tk_foreign
        };
```

TypeCode **TypeCodeNew** (TCKind tag, ...); † — Creates a new TypeCode instance.

† Takes *no* implicit parameters.

TypeCodeNew (tk_objref, string interfaceId);
TypeCodeNew (tk_string, long maxLength);
TypeCodeNew (tk_sequence, TypeCode seqTC, long maxLength);
TypeCodeNew (tk_array, TypeCode arrayTC, long length);
TypeCodeNew (tk_pointer, TypeCode ptrTC);
TypeCodeNew (tk_self, string structOrUnionName);
TypeCodeNew (tk_foreign, string typename, string impCtx, long instSize);

TypeCodeNew (tk_struct,
string name,
string mbrName,
TypeCode mbrTC, [...,]
[mbrName and mbrTC repeat as needed]
NULL);

TypeCodeNew (tk_union,
string name,
TypeCode swTC,
long flag,
long labelValue,
string mbrName,
TypeCode mbrTC, [...,]
[flag, labelValue, mbrName and mbrTC repeat as needed]
NULL);

TypeCodeNew (tk_enum,
string name,
string enumId, [...,]
[enumIds repeat as needed]
NULL);

TypeCodeNew (TCKind allOtherTagValues);

'allOtherTagValues' represents one of the values:
tk_null, tk_void, tk_short, tk_long,
tk_ushort, tk_ulong, tk_float, tk_double,
tk_boolean, tk_char, tk_octet,
tk_any, tk_TypeCode, or tk_Principal
All of these tags represent basic IDL data types that do not require any other descriptive parameters.

long **TypeCode_param_count** ( ); — Obtains the number of parameters available in a given TypeCode.

any **TypeCode_parameter** (long index); — Obtains a specified parameter from a given TypeCode.

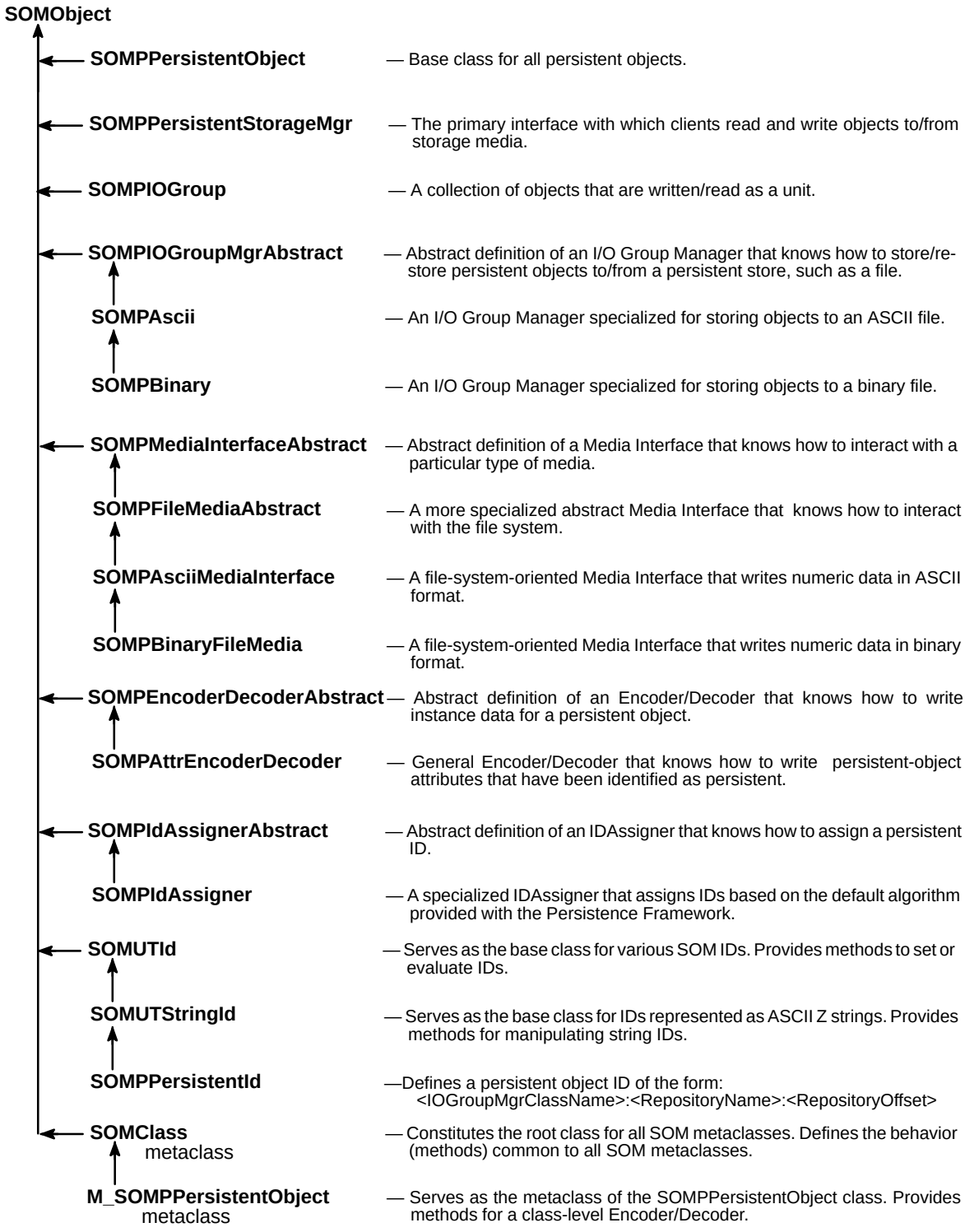
void **TypeCode_print** ( ); — Writes all of the information contained in a given TypeCode to "stdout".

void **TypeCode_setAlignment** (short alignment); — Overrides the default alignment associated with the given TypeCode.

long **TypeCode_size** ( ); — Provides the minimum size of an instance of the abstract data type described by a given TypeCode.

Persistence Framework Quick Reference

Class Organization



M_SOMPPersistentObject class

(see po.idl)

string **sompGetClassLevelEncoderDecoderName**();	Gets the class-level default Encoder/Decoder class name.
void **sompSetClassLevelEncoderDecoderName**(in string myName);	Sets the class-level default Encoder/Decoder class name.

SOMPAsciiMediaInterface class

(see fmi.idl)

string **sompGetMediaName** (in string toBuffer);	Gets the name of the file for the current media interface object.
void **sompInitSpecific** (inout mediaInfoType mediaInfo);	Initializes the Media Interface permissions and creation flags.
unsigned long **sompQueryBlockSize** ( );	Returns an optimal block size for I/O operations.
void **sompStat** (inout sompstat fileStats);	Sets sompstat information for the file associated with the current object.

SOMPEncoderDecoderAbstract class

(see eda.idl)

void **sompEDRead** (in SOMPMediaInterfaceAbstract thisMedia, in SOMPPersistentObject thisObject);	Reads the stored instance data of a persistent object, and updates the persistent object to reflect the new data.
void **sompEDWrite** (in SOMPMediaInterfaceAbstract thisMedia, in SOMPPersistentObject thisObject);	Writes the instance data of a persistent object to storage media in a format that is compatible with the sompEDRead method.

SOMPFileMediaAbstract class

(see fma.idl)

long **sompGetOffset** ( );	Returns the current offset within the Media Interface file.
void **sompInitReadOnly** (in string mediaInfo);	Initializes the Media Interface for read-only access.
void **sompInitReadWrite** (in string mediaInfo);	Initializes the Media Interface for read/write access.
void **sompReadBytes** (in string bytestream, in long length);	Reads a byte stream of the specified length from the current position in the Media Interface file into a preallocated buffer.
void **sompReadCharacter** (inout char c);	Reads a character from the Media Interface file.
void **sompReadDouble** (inout double f);	Reads double-precision floating-point data from the Media Interface file.
void **sompReadFloat** (inout float f);	Reads floating-point data from the Media Interface file.

void **sompReadLine** (
in string buffer,
in long bufsize);

Reads a line from the Media Interface file into a preallocated buffer.

void **sompReadLong** (
inout long i2);

Reads a long integer from the Media Interface file.

void **sompReadOctet** (
inout octet f);

Reads an eight-bit quantity from the Media Interface file.

void **sompReadShort** (
inout short i1);

Reads a short integer from the Media Interface file.

void **sompReadSomobject** (
inout SOMObject so);

Reads a SOM Object from the Media Interface file. If the object is persistent, it will be read; if the object is non-persistent, the proper class is just instantiated.

void **sompReadString** (
inout string rstring);

Allocates space for a string and reads the string from the Media Interface file.

void **sompReadStringToBuffer** (
in string buffer,
in long bufsize);

Reads a string into a preallocated buffer (as a null-terminated string) for error checking. If the string exceeds the bufsize, it will be truncated.

void **sompReadTypeCode** (
inout TypeCode tc);

Reads an IDL type code from a Media Interface file.

void **sompReadUnsignedLong** (
inout ulong i2);

Reads an unsigned long integer from the Media Interface file.

void **sompReadUnsignedShort** (
inout ushort i1);

Reads an unsigned short integer from the Media Interface file.

void **sompSeekPosition** (
in long offset);

Positions the Media Interface file to a specified offset.

void **sompSeekPositionRel** (
in long offset);

Positions the Media Interface file to a specified offset, relative to the current position in the file.

void **sompWriteBytes** (
in string bytestream
in long length);

Writes a byte stream of a specified length to the Media Interface file.

void **sompWriteCharacter** (
in char c);

Writes a character to the Media Interface file.

void **sompWriteDouble** (
in double f);

Writes double-precision floating-point data to the Media Interface file.

void **sompWriteFloat** (
in float f);

Writes floating-point data to the Media Interface file.

void **sompWriteLine** (
in string buffer);

Writes a line to the Media Interface file.

void **sompWriteLong** (
in long i2);

Writes a long integer to the Media Interface file.

void **sompWriteOctet** (
in octet f);

Writes an eight-bit quantity to the Media Interface file.

void **sompWriteShort** (
in short i1);

Writes a short integer to the Media Interface file.

void **sompWriteSomobject** (
in SOMObject so,
in SOMObject parentObject);

Writes a SOM Object to the Media Interface file. If the object is persistent, it will be stored; if the object is non-persistent, just its class name will be stored.

void **sompWriteString** (
in string string);

Writes a string to the Media Interface file.

void **sompWriteTypeCode** (
in TypeCode tc);

Writes a restorable version of an IDL type code to the Media Interface file.

void **sompWriteUnsignedLong** (
in ulong i2);

Writes an unsigned long integer to the Media Interface file.

void **sompWriteUnsignedShort** (
in ushort i1);

Writes an unsigned short integer to the Media Interface file.

SOMPIdAssignerAbstract class

(see poida.idl)

void **sompGetSystemAssignedId** (
inout SOMPPersistentId id);

Assigns a persistent ID to a persistent object.

SOMPIOGroup class

(see iogrp.idl)

void **sompAddToGroup** (
in SOMObject newObject,
in SOMPIOGroupKey key);

Adds a specified object to a given I/O Group of objects that are written/read as a unit.

long **sompCount** ( );

Determines the number of objects composing an I/O Group that is written/read as a unit.

SOMObject **sompFindByKey** (
in SOMPIOGroupKey key);

Gets the object that corresponds to a specified key.

SOMObject **sompFirst** ( );

Gets the first available object in an I/O Group.

void **sompFreeIterator** (
in SOMPIteratorHandle iteratorHandle);

Frees the resources used by a given iterator for a designated I/O Group.

SOMPIteratorHandle **sompNewIterator** ( );

Gets a handle for use in iterating through an I/O Group of objects that are written/read as a unit.

SOMObject **sompNextObjectInGroup** (
in SOMPIteratorHandle iteratorHandle);

Gets the next object in an I/O Group of objects that are written/read as a unit.

SOMObject **sompRemoveFromGroup** (
in SOMPIOGroupKey key);

Removes and returns a designed object that was part of an I/O Group.

SOMPIOGroupMgrAbstract class

(see iogma.idl)

void **sompDeleteObjectFromGroup** (
in SOMPPersistentId objectID);

Deletes a given object from a specified group in persistent storage.

void **sompFreeMediaInterface** ( );

Frees a specified I/O Group Manager's interface to storage media.

SOMPMediaInterfaceAbstract **sompGetMediaInterface** ( );	Obtains a specified I/O group's media interface.
boolean **sompGroupExists** (in string IOInfo);	Determines whether an I/O group exists in persistent storage.
SOMPMediaInterfaceAbstract **sompInstantiateMediaInterface** ( );	Instantiates the proper file media interface used by this I/O Group Manager.
boolean **sompMediaFormatOk** (in string mediaFormatName);	Checks whether a media format is supported by this I/O Group Manager.
void **sompNewMediaInterface** (in string IOInfo);	Initializes a new Media Interface for an I/O Group Manager and prepares it for I/O.
boolean **sompObjectInGroup** (in SOMPPersistentId objectId);	Checks whether a given object exists in a given group in persistent storage.
SOMPIOGroup **sompReadGroup** (in SOMPPersistentId objectId);	Instantiates and initializes the I/O Group of a given persistent object. Persistent data of each object is not read, and the objects are left in an unstable state.
void **sompReadObjectData** (in SOMPPersistentObject thisPo);	Reads a persistent object from storage, effectively stabilizing an unstable object.
boolean **sompWriteGroup** (in SOMPPersistentObject thisPo);	Stores all of the dirty objects in an I/O Group or an individual object, as determined by the I/O Group Mgr implementor.

SOMPMediaInterfaceAbstract class

(see mia.idl)

void **sompClose** ( );	Closes the Media Interface.
void **sompOpen** ( );	Opens the Media Interface.

SOMPPersistentId class

(see pid.idl)

boolean **sompEqualsIOGroupName** (in SOMPPersistentId id);	Compares two persistent IDs to determine if they are the same.
long **sompGetGroupOffset** ( );	Gets the offset portion (the RepositoryOffset) of a persistent ID that identifies the storage location of a persistent object.
string **sompGetIOGroupMgrClassName** (in string toBuffer);	Gets the "I/O Group Manager class" portion of a persistent ID that identifies the storage location of a persistent object.
short **sompGetIOGroupMgrClassNameLen** ( );	Gets the length of the "I/O Group Manager class name" portion of a persistent ID that identifies the storage location of a persistent object.
string **sompGetIOGroupName** (in string toBuffer);	Gets the "I/O Group name" portion (the RepositoryName) of a persistent ID that identifies the storage location of a persistent object.

short **sompGetIOGroupNameLen** ( );	Gets the length of the "I/O Group name" portion (the RepositoryName) of a persistent ID that identifies the storage location of a persistent object.
void **sompSetGroupOffset** (in long offset);	Sets the offset portion (the RepositoryOffset) of a persistent ID that will identify the storage location of a persistent object.
void **sompSetIOGroupMgrClassName** (in string newName);	Sets the "I/O Group Manager class" portion of a persistent ID that will identify the storage location of a persistent object.
void **sompSetIOGroupName** (in string newName);	Sets the "I/O Group name" portion (the RepositoryName) of a persistent ID that will identify the storage location of a persistent object.

SOMPPersistentObject class

(see po.idl)

void **sompActivated** ( );	Notifies a persistent object that it's persistent data has just been read. This method is invoked by the Persistence Framework.
boolean **sompCheckState** (in ushort state);	Checks whether an object is in a specified state.
void **sompClearState** (in ushort state);	Clears the specified state.
boolean **sompEquals** (in SOMPPersistentId otherId);	Checks whether an object's ID is equivalent to a specified ID.
void **sompFreeEncoderDecoder** ( );	Frees a persistent object's Encoder/Decoder.
boolean **sompGetDirty** ( );	Determines whether the state of an object is "dirty."
SOMPEncoderDecoderAbstract **sompGetEncoderDecoder** ( );	Returns an Encoder/Decoder.
string **sompGetEncoderDecoderName** ( );	Gets the class name of the appropriate Encoder/Decoder.
SOMPIOGroup **sompGetIOGroup** ( );	Gets a pointer to the SOMPIOGroup object for a given persistent object.
SOMPPersistentId **sompGetPersistentId** ( );	Gets the ID for a persistent object, returning it as a persistent ID.
string **sompGetPersistentIdString** (in string outBuf);	Gets the persistent ID for a persistent object, returning it as a string.
string **sompGetRelativeIdString** (in SOMObject ifNearThisObj, in string relativeIdString);	Constructs a relative persistent ID for a persistent object, returning it as a string.
void **sompInitGivenId** (in SOMPPersistentId thisId);	Copies information from an ID parameter into the object's persistent ID.
SOMPIOGroup **sompInitIOGroup** (in SOMObject nearPersistentObj);	Initializes the I/O group of a specified persistent object.

void **sompInitNearObject** (in SOMPPersistentObject nearObj);	Gives a specified persistent object a persistent ID nearby the ID of another specified persistent object.
void **sompInitNextAvail** (in SOMPIdAssigner thisAssigner);	Gives a specified persistent object the next available persistent ID, as determined by the IdAssigner.
boolean **sompIsDirty** ( );	Tells whether an object should be considered dirty.
void **sompMarkForCompaction** ( );	Marks a persistent object's I/O Group for compaction the next time the object is saved.
void **sompPassivate** ( );	Notifies a persistent object that its persistent data is about to be stored. This method is invoked by the Persistence Framework.
void **sompSetDirty** ( );	Sets the state of a persistent object to "dirty."
void **sompSetEncoderDecoderName** (in string myName);	Binds an Encoder/Decoder class name to a persistent object.
void **sompSetState** (in ushort state);	Sets a persistent object to a specified state.

SOMPPersistentStorageMgr class

(see psma.idl)

void **sompAddIdToReadSet** (in SOMPPersistentId objectId);	Adds the specified persistent ID to the set of IDs that represent persistent objects to be restored by the sompRestoreObject method as part of this read request**.**
void **sompAddObjectToWriteSet** (in SOMPPersistentObject thisObject);	Adds the specified persistent object to the set of objects to be written by the method sompStoreObject, as part of this write request.
void **sompDeleteObject** (in SOMPPersistentId objectID);	Deletes a specified persistent object from persistent storage.
boolean **sompObjectExists** (in SOMPPersistentId objectID);	Determines whether a persistent object exists in persistent storage.
SOMObject **sompRestoreObject** (in SOMPPersistentId objectID);	Restores a persistent object and all its contained persistent objects to a stable state.
SOMObject **sompRestoreObjectFromIdString**(in string idString);	Restores a persistent object and all its persistent children to a stable state, given the object's string ID.
SOMObject **sompRestoreObjectWithoutChildren** (in SOMPPersistentId objectID);	Restores a persistent object to a stable state. Contained objects are restored only in an unstable state.
void **sompStoreObject** (in SOMPPersistentObject thisObject);	Stores a persistent object and all its persistent children.
void **sompStoreObjectWithoutChildren** (in SOMPPersistentObject thisObject);	Stores a persistent object, but does not store contained objects.

SOMUTId class

(see somida.idl)

short **somutCompareId** (
in SOMUTId otherId);
Performs an ordered comparison of one ID with another.

boolean **somutEqualsId** (
in SOMUTId otherId);
Determines whether an ID is equal to the given ID.

ulong **somutHashId** ( );
Determines the hash value equivalent to a given ID.

void **somutSetIdId** (
in SOMUTId otherId);
Sets an ID equal to a given ID.

SOMUTStringId class

(see somsid.idl)

short **somutCompareString** (
in string IdString);
Performs an ordered comparison of an ID with a string.

boolean **somutEqualsString** (
in string idString);
Determines whether an ID is equal to the given string.

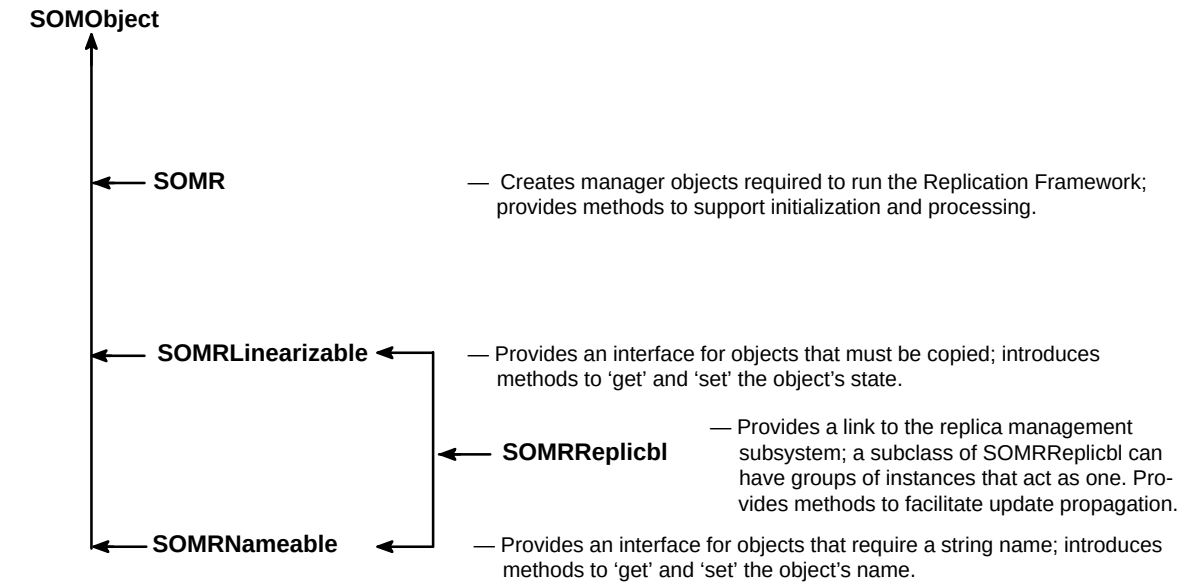
string **somutGetIdString** (
in string toBuffer);
Converts an ID to a string.

long **somutGetIdStringLen** ( );
Returns the length of an ID string.

long **somutSetIdString** (
in string IdString);
Sets an ID equal to a given string.

Replication Framework Quick Reference

Class Organization



SOMRLinearizable class

(see linear.idl)

void **somrGetState** (inout string buf);	Converts the internal state of an object into a byte string.
void **somrSetState** (in string buf);	Converts a given linear byte string into its internal state.

SOMRNameable class

(see nameable.idl)

string **somrGetObjName** ( );	Returns a pointer to an object's name string.
void **somrSetObjName** (in string name);	Sets the name of a nameable object.

SOMRReplicbl class

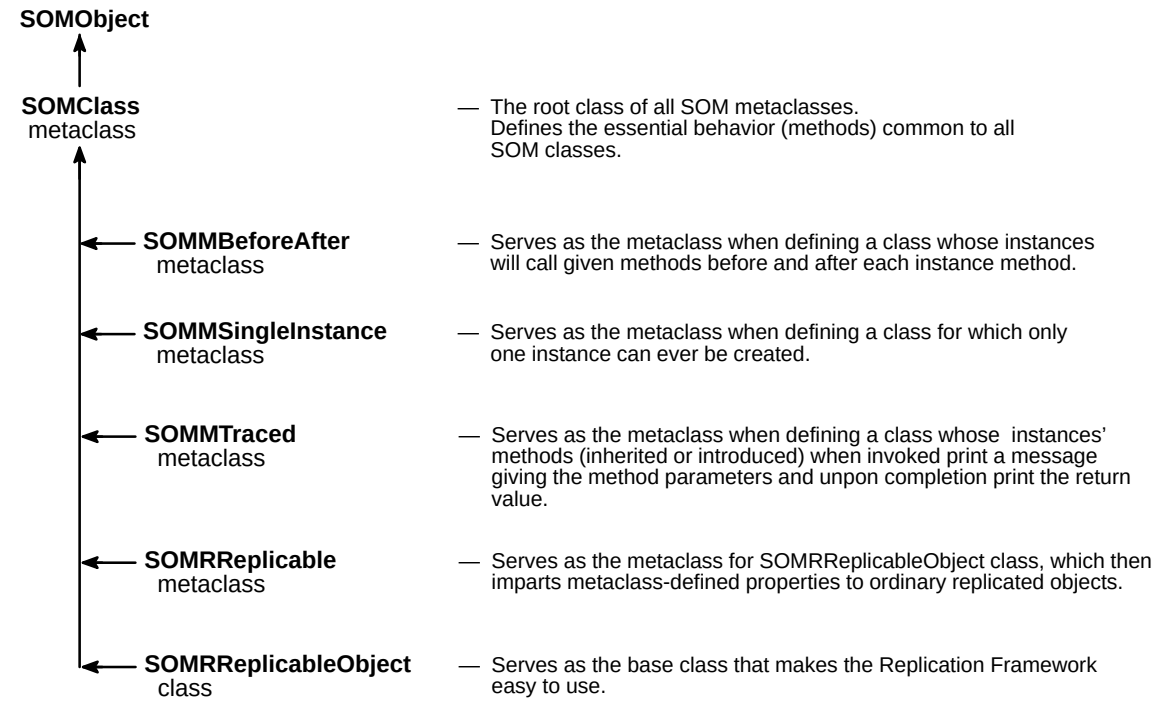
(see replicbl.idl)

void **somrApplyUpdates** (string buffer, int bufferLen, int objIntId);	Interprets the buffer received as an update to its state.
void **somrDoDirective** (in string str);	Interprets a directive sent to a replica.
long **somrGetSecurityPolicy** ( );	Returns the security policy for replicated objects.

void **somrLock** ( );	Gets a lock on the replica of the object when doing value logging.
void **somrLockNlogOp** (in string classname, in string methodname, in va_list *ap);	Gets a lock on the replica of the object and logs the method.
void **somrPin** ( );	Pins the lock to this replica until somrUnPin is called.
void **somrReleaseLockNAbortOp** ( );	Aborts the operation begun by calling the somrLockNLogOp method.
void **somrReleaseLockNAbortUpdate** ( );	Aborts the operation begun by calling somrLock.
void **somrReleaseNPropagateOperation** ( );	Releases the lock and propagates the operation log.
void **somrReleaseNPropagateUpdate** (in string clsname, in string buffer, in int bufferlen, in int intObjId);	Requests the release of the lock and propagates the value of the replica.
long **somrRepInit** (in char lType, in char mode);	Makes the object ready for replication.
void **somrRepUninit** ( );	Destroys the setup for replication.
void **somrUnPin** ( );	Unpins the lock so that it can be obtained by another replica.

Metaclass Framework Quick Reference

Class Organization



SOMMBeforeAfter metaclass

(see sombacls.idl)

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

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

boolean **sommBeforeMethod** (
in SOMObject object,
in somId methodID,
in va_list ap);

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

SOMMSingleInstance metaclass

(see snglicls.idl)

SOMObject **sommGetSingleInstance** ( );

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

SOMMTraced metaclass

(see somtrcls.idl)

attribute boolean **sommTraceIsOn**

Indicates whether or not tracing is turned on for a class, giving dynamic control over the trace facility.

SOMRReplicableObject class

(see somrcls.idl)

char **somrLoggingType** ( );

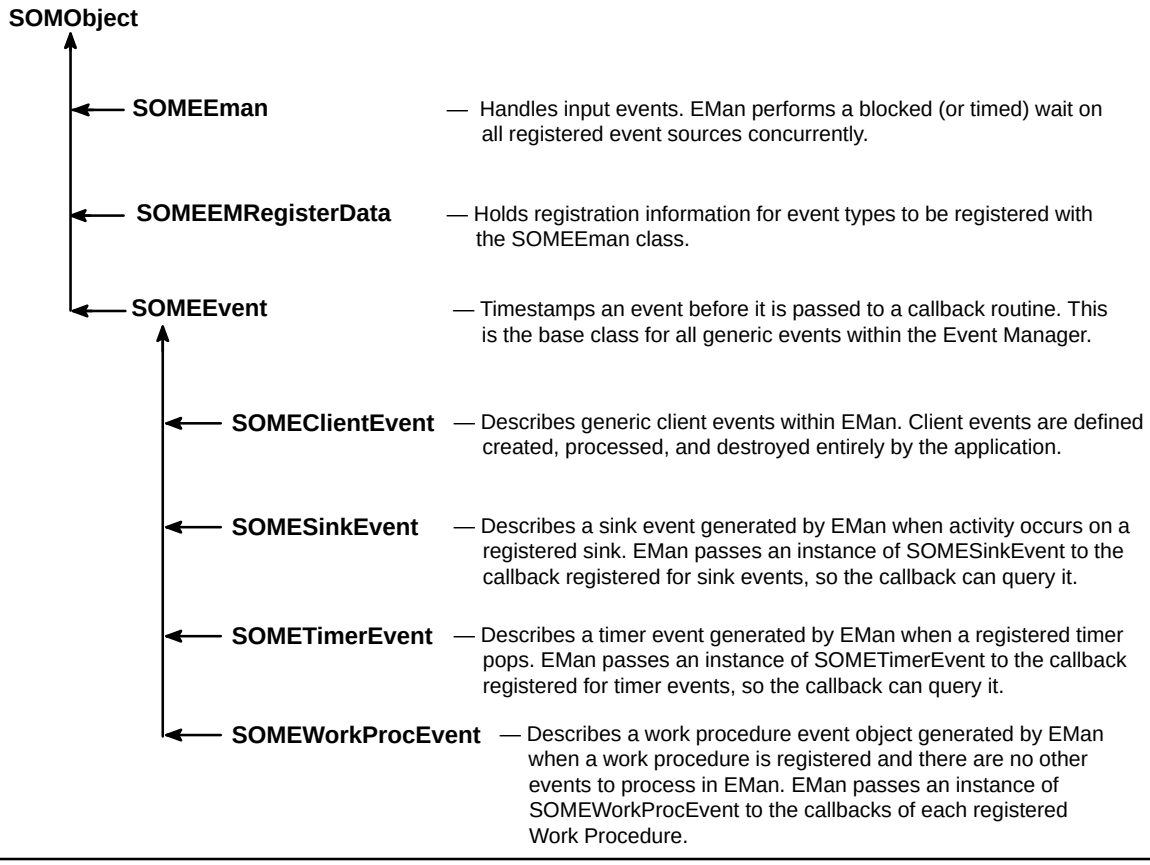
Enables querying of the logging type for a replicable object.

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

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

Event Management Framework Quick Reference

Class Organization



SOMEClientEvent class

(see clientev.idl)

void* **somevGetEventClientData** ( );	Returns the user-defined data associated with a client event.
string **somevGetEventClientType** ( );	Returns the type name of a client event.
void **somevSetEventClientData** (in void* clientData);	Sets the user-defined data of a client event.
void **somevSetEventClientType** (in string clientType);	Sets the type name of a client event.

SOMEEMan class

(see eman.idl)

void **someChangeRegData** (in long registrationId, in SOMEEMRegisterData registerData);	Changes the registration data associated with a specified registration ID.

void **someGetEManSem** ( );	Acquires EMan semaphore(s) to achieve mutual exclusion with EMan's activity.
void **someProcessEvent** (in unsigned long mask);	Processes one event.
void **someProcessEvents** ( );	Processes infinite events.
void **someQueueEvent** (in SOMEClientEvent event);	Enqueues the specified client event.
long **someRegister** (in SOMEEMRegisterData registerData, in SOMObject targetObject, in string targetMethod, in void *targetData);	Registers an object/method pair with EMan, given a specified registerData object.
long **someRegisterEv** (in SOMEEMRegisterData registerData, in SOMObject targetObject, inout Environment callbackEv, in string targetMethod, in void *targetData);	Registers the (object, method, Environment parameter) combination of a callback with EMan, given a specified registerData object.
long **someRegisterProc** (in SOMEEMRegisterData registerData, in EMRegProc *targetProcedure, in void *targetData);	Register the procedure with EMan given the specified registerData.
void **someReleaseEManSem** ( );	Releases the semaphore obtained by the someGetEManSem method.
void **someShutdown** ( );	Shuts down an EMan event loop. (That is, this makes the someProcessEvents return!)
void **someUnRegister** (in long registrationId);	Unregisters the event interest associated with a specified registrationId within EMan.

SOMEEMRegisterData class

(see emregdat.idl)

void **someClearRegData** ( );	Clears the registration data.
void **someSetRegDataClientType** (in string clientType);	Sets the type name for a client event.
void **someSetRegDataEventMask** (in long eventType, in va_list ap);	Sets the generic event mask within the registration data using NULL terminated event type list.
void **someSetRegDataSink** (in long sink);	Sets the file descriptor (or socket ID, or message queue ID) for the sink event.
void **someSetRegDataSinkMask** (in unsigned long sinkmask);	Sets the sink mask within the registration data object.
void **someSetRegDataTimerCount** (in long count);	Sets the number of times the timer will trigger, within the registration data.
void **someSetRegDataTimerInterval** (in long interval);	Sets the timer interval within the registration data.

SOMEEvent class

(see event.idl)

unsigned long **somevGetEventTime** ( );
Returns the time of the generic event in milliseconds.

unsigned long **somevGetEventType** ( );
Returns the type of the generic event.

void **somevSetEventTime** (
in unsigned long time);
Sets the time of the generic event (time is in milliseconds).

void **somevSetEventType** (
in unsigned long type);
Sets the type of the generic event.

SOMESinkEvent class

(see sinkev.idl)

long **somevGetEventSink** ( );
Returns the sink, or source of I/O, of the generic sink event.

void **somevSetEventSink** (
in long sink);
Sets the sink, or source of I/O, of the generic sink event.

SOMETimerEvent class

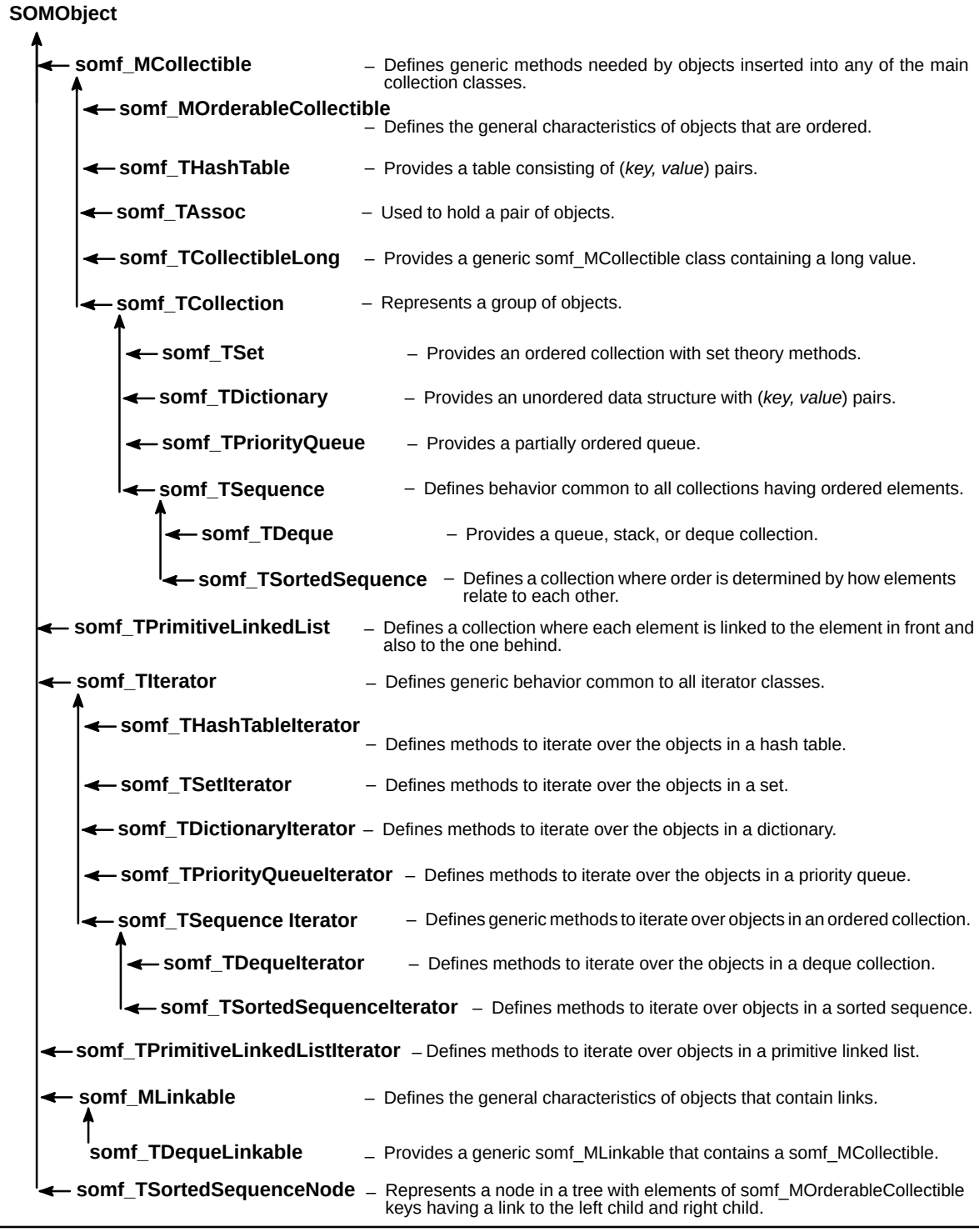
(see timerev.idl)

void **somevGetEventInterval** ( );
Returns the interval of the generic timer event (time in milliseconds).

void **somevSetEventInterval** (
in long interval);
Sets the interval of the generic timer event (in milliseconds).

Collection Classes Quick Reference

Organization of Collection Classes



somf_MCollectible class

(see mcollect.idl)

somf_MCollectible **somfClone** ( );	Provides a general polymorphic duplication operation.
somf_MCollectible **somfClonePointer** (in somf_MCollectible clonee);	Returns a pointer to a Clone.
long **somfHash** ( );	Returns a value suitable for use as a hashing probe for the receiving object.
boolean **somfIsEqual** (in somf_MCollectible obj);	Returns TRUE if a given obj is isomorphic to the receiving object.
boolean **somfIsNotEqual** (in somf_MCollectible obj);	Returns TRUE if a specified obj is not isomorphic to the receiving object.
boolean **somfIsSame** (in somf_MCollectible obj);	Performs a pointer comparison between the receiving object and another specified object, obj.

somf_MLinkable class

(see mlink.idl)

somf_MLinkable **somfGetNext** ( );	Gets a pointer to the next somf_MLinkable object.
somf_MLinkable **somfGetPrevious** ( );	Gets a pointer to the previous somf_MLinkable object.
somf_MLinkable **somfMLinkableInit** (in somf_MLinkable n, in somf_MLinkable p);	Initializes a new somf_MLinkable object, given pointers to its next and previous objects.
void **somfSetNext** (in somf_MLinkable aLink);	Sets a link pointer to the next somf_MLinkable object, given a pointer to the object that should come after the receiving object.
void **somfSetPrevious** (in somf_MLinkable aLink);	Sets a link pointer to the previous somf_MLinkable object, given a pointer to the object that should come before the receiving object.

somf_MOrderableCollectible class

(see morder.idl)

EComparisonResult **somfCompare** (in somf_MOrderableCollectible obj);	Compares a specified obj to the receiving object, and returns a value indicating obj's comparative size.
boolean **somfIsGreaterThan** (in somf_MOrderableCollectible obj);	Compares two objects and returns TRUE if a given obj is "greater than" the receiving object.
boolean **somfIsGreaterThanOrEqualTo** (in somf_MOrderableCollectible obj);	Compares two objects and returns TRUE if a specified obj is "greater than" or "equal to" the receiving object.
boolean **somfIsLessThan** (in somf_MOrderableCollectible obj);	Compares two objects and returns TRUE if a given obj is "less than" the receiving object.
boolean **somfIsLessThanOrEqualTo** (in somf_MOrderableCollectible obj);	Compares two objects and returns TRUE if a given obj is "less than" or "equal to" the receiving object.

somf_TAssoc class

(see tassoc.idl)

somf_MCollectible **somfGetKey** ( );
Gets the key of an associated (key, value) pair.

somf_MCollectible **somfGetValue** ( );
Gets the value to an associated (key, value) pair.

void **somfSetKey** (
in somf_MCollectible k);
Sets the key of an associated (key, value) pair.

void **somfSetValue** (
in somf_MCollectible v);
Sets the value of an associated (key, value) pair.

somf_TAssoc **somfTAssocInitM** (
in somf_MCollectible k);
Initializes a somf_TAssoc object to a given key (k). The value (v) is set to SOMF_NIL.

somf_TAssoc **somfTAssocInitMM** (
in somf_MCollectible k,
in somf_MCollectible v);
Initializes a somf_TAssoc object to a given key (k) and value (v).

somf_TCollectibleLong class

(see tclong.idl)

long **somfGetValue** ( );
Gets the value of the long in the receiving object.

long **somfHash** ( );
Returns a value suitable for use as a hashing probe for the receiving object. (Actually, it returns the value of the long.)

boolean **somfIsEqual** (
in somf_MCollectible obj);
Compares two objects and returns TRUE if a given obj is isomorphic to the receiving object.

void **somfSetValue** (
in long v);
Sets the value of a long in a somf_TCollectibleLong object.

somf_TCollectibleLong
somfTCollectibleLongInit (
in long v);
Initializes a new object of class somf_TCollectibleLong.

somf_TCollection class

(see tcollect.idl)

somf_MCollectible **somfAdd** (
in somf_MCollectible obj);
Adds a specified obj to a collection.

void **somfAddAll** (
in somf_TCollection col);
Adds all of the objects from a given collection into the receiving object.

long **somfCount** ( );
Gets the number of objects in a collection.

somf_TIterator **somfCreateIterator** ( );
Returns a new iterator that is suitable for iterating over the objects in this collection.

void **somfDeleteAll** ( );
Removes all of the objects from the receiving object and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.)

boolean **somfIsEqual** (
in somf_MCollectible obj);
Compares two objects and returns TRUE if a specified obj is isomorphic to the receiving object.

somf_MCollectible **somfMember** (
in somf_MCollectible obj);
Gets an obj in the collection.

somf_MCollectible **somfRemove** (in somf_MCollectible obj);	Removes an object from a collection.
void **somfRemoveAll** ( );	Removes all of the objects in a collection.
void **somfSetTestFunction** (in somf_MCollectibleCompareFn testfn);	Sets the test function for a collection.
somf_TCollection **somfTCollectionInit** (in somf_MCollectibleCompareFn testfn);	Initializes a new object of class somf_TCollection.
somf_MCollectibleCompareFn **somfTestFunction** ( );	Determines the function that a collection uses for comparison testing.

somf_TDeque class

(see tdeq.idl)

somf_MCollectible **somfAdd** (in somf_MCollectible obj);	Adds an object to a deque collection.
void **somfAddAfter** (in somf_MCollectible existingobj, in somf_MCollectible tobeadded);	Adds a new object to a deque collection after a specified existing object.
void **somfAddBefore** (in somf_MCollectible existobj, in somf_MCollectible tobeadded);	Adds a new object to a deque collection before a specified existing object.
void **somfAddFirst** (in somf_MCollectible obj);	Adds a new object as the first object in a deque collection.
void **somfAddLast** (in somf_MCollectible obj);	Adds a new object as the last object in a deque collection.
somf_MCollectible **somfAfter** (in somf_MCollectible obj);	Gets the object found after a specified object in a deque collection.
void **somfAssign** (in somf_TDeque s);	Assigns a deque collection as being equal to a given source deque.
somf_MCollectible **somfBefore** (in somf_MCollectible obj);	Gets the object found before a specified object in a deque collection.
long **somfCount** ( );	Gets the number of objects in a deque collection.
somf_TIterator **somfCreateIterator** ( );	Returns a new iterator that is suitable for iterating over the objects in this deque collection.
somf_TDequeLinkable **somfCreateNewLink** (in somf_TDequeLinkable p, n somf_TDequeLinkable n, n somf_MCollectible v);	Creates a new link in a somf_TDeque collection, given two objects as the previous and next somf_TDequeLinkable links, and the value of the new link.
somf_TSequenceIterator **somfCreateSequenceIterator** ( );	Returns a new iterator that is suitable for iterating over the objects in the given deque collection.
void **somfDeleteAll** ( );	Removes all of the objects from a deque collection and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.)
somf_MCollectible **somfFirst** ( );	Gets the first object in a deque collection.

void **somfInsert** (in somf_MCollectible obj);	Adds an object to the end of the deque/queue.
somf_MCollectible **somfLast** ( );	Gets the last object in a given deque collection.
somf_MCollectible **somfMember** (in somf_MCollectible obj);	Gets an object in a deque collection.
somf_MCollectible **somfPop** ( );	Removes the object on the top of a deque/stack.
void **somfPush** (in somf_MCollectible obj);	Adds an object to the top of a deque/stack.
somf_MCollectible **somfRemove** (in somf_MCollectible obj);	Removes an object from a deque collection.
void **somfRemoveAll** ( );	Removes all of the objects from a deque collection.
somf_MCollectible **somfRemoveFirst** ( );	Removes the first object in a deque collection.
somf_MCollectible **somfRemoveLast** ( );	Removes the last object in a deque collection.
somf_MCollectible **somfRemoveQ** ( );	Removes the first object from a deque/queue.
somf_TDeque **somfTDequeInitD** (in somf_TDeque s);	Initializes a new deque, setting it equal to a given somf_TDeque source object.
somf_TDeque **somfTDequeInitF** (in somf_MCollectibleCompareFn testfn);	Initializes a new deque collection, specifying the comparison function that it will use.

somf_TDequeIterator class

(see tdeqitr.idl)

somf_MCollectible **somfFirst** ( );	Resets the iterator and returns the first object from a deque collection.
somf_MCollectible **somfLast** ( );	Gets the last object in the deque collection.
somf_MCollectible **somfNext** ( );	Gets the next object in a deque collection.
somf_MCollectible **somfPrevious** ( );	Gets the previous object in a deque collection.
void **somfRemove** ( );	Removes the current object from a deque collection.
somf_TDequeIterator **somfTDequeIteratorInit** (in somf_TDeque h);	Initializes a somf_TDequeIterator iterator for a deque collection.

somf_TDequeLinkable class

(see tdeqlink.idl)

somf_MCollectible **somfGetValue** ( );	Gets the value from a somf_TDequeLinkable node.
void **somfSetValue** (in somf_MCollectible v);	Sets the value of a given somf_TDequeLinkable node.
somf_TDequeLinkable **somfTDequeLinkableInitDD** (in somf_TDequeLinkable previous, in somf_TDequeLinkable next);	Initializes a new somf_TDequeLinkable node, by specifying the adjacent nodes to which it will link. This method does not set a value for the node.
somf_TDequeLinkable **somfTDequeLinkableInitDDM** (in somf_TDequeLinkable previous, in somf_TDequeLinkable next, in somf_MCollectible value);	Initializes a new somf_TDequeLinkable node. This includes specifying the adjacent nodes and setting the value of the node.

somf_TDictionary class

(see tdict.idl)

somf_MCollectible **somfAdd** (in somf_MCollectible obj);	Adds a specified obj (representing a key, value pair) to the dictionary.
somf_MCollectible **somfAddKeyValuePairMM** (in somf_MCollectible key, in somf_MCollectible val);	Adds a (key, value) pair to the receiving dictionary object, and returns a removed object (if removal was necessary).
somf_MCollectible **somfAddKeyValuePairMMB** (in somf_MCollectible key, in somf_MCollectible val, in boolean replace);	Adds a (key, value) pair to a dictionary, unless the Boolean argument prohibits replacement of a conflicting pair.
void **somfAssign** (in somf_TDictionary source);	Assigns a dictionary as being “equal” to a given source dictionary.
somf_THashTable **somfCopyImplementation** ( );	Returns a hash table that is a copy of the hash table in a given dictionary.
long **somfCount** ( );	Gets the number of objects in a dictionary.
somf_TIterator **somfCreateIterator** ( );	Returns a new iterator that is suitable for iterating over the objects in this dictionary.
somf_THashTable **somfCreateNewImplementationF** (in somf_MCollectibleCompareFn testfn);	Creates a new hash table for a dictionary, given its comparison test function.
somf_THashTable **somfCreateNewImplementationFL** (in somf_MCollectibleCompareFn testfn, in long tablesize);	Creates a new hash table for a dictionary, given its comparison test function and its initial table size.
somf_THashTable **somfCreateNewImplementationFLL** (in somf_MCollectibleCompareFn testfn, in long tablesize, in long rate);	Creates a new hash table for a dictionary, given its comparison test function, the initial table size, and the table’s growth rate.
somf_THashTable **somfCreateNewImplementationFLLL** (in somf_MCollectibleCompareFn testfn, in long tablesize, in long rate, in long threshold);	Creates a new hash table for a dictionary, given its comparison test function, the initial table size, the table’s growth rate, and the table’s rehash threshold.
void **somfDeleteAll** ( );	Removes all of the (key, value) pairs from a dictionary and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the dictionary.)
void **somfDeleteAllKeys** ( );	Removes all of the (key, value) pairs from a dictionary. The procedure resets the count to zero and calls the destructor on every key in the dictionary.
void **somfDeleteAllValues** ( );	Removes all of the (key, value) pairs from a dictionary. The procedure resets the count to zero and calls the destructor on every value in the hash table.

somf_MCollectible **somfDeleteKey** (in somf_MCollectible key);	Deletes a specified key from the associated (key, value) pair, and removes the (key, value) pair from the dictionary.
somf_MCollectibleHashFn **somfGetHashFunction** ( );	Returns a pointer to the hash function used by a given dictionary.
somf_MCollectible **somfKeyAtM** (in somf_MCollectible val);	Gets a dictionary's first key that has a specified val as its associated value. Note: This method involves a slow search.
somf_MCollectible **somfKeyAtMF** (in somf_MCollectible val, in somf_MCollectibleCompareFn testfn);	Gets a dictionary's first key that has a specified val as its associated value. The method includes an argument specifying the function to be used for comparing the values. Note: This method involves a slow search.
somf_MCollectible **somfMember** (in somf_MCollectible obj);	Gets a specified (key, value) pair in the dictionary, if it is found.
somf_MCollectible **somfRemove** (in somf_MCollectible key);	Removes from the dictionary the (key, value) pair associated with a given key.
void **somfRemoveAll** ( );	Removes all of the (key, value) pairs from a dictionary.
void **somfSetHashFunction** (in somf_MCollectibleHashFn fn);	Sets a dictionary's hash-function pointer to a given function.
somf_TDictionary **somfTDictionaryInitD** (in somf_TDictionary dictionary);	Initializes a new dictionary, setting it equal to another specified dictionary.
somf_TDictionary **somfTDictionaryInitF** (in somf_MCollectibleCompareFn testfn);	Initializes a new dictionary, given its comparison test function.
somf_TDictionary **somfTDictionaryInitFL** (in somf_MCollectibleCompareFn testfn, in long sizeHint);	Initializes a new dictionary, given its comparison test function and its initial size.
somf_TDictionary **somfTDictionaryInitFLL** (in somf_MCollectibleCompareFn testfn, in long sizeHint, in long growthRate);	Initializes a new dictionary, given its comparison test function, its initial size, and its initial growth rate. Note: This method is equivalent to the somfTDictionaryInitLLF method.
somf_TDictionary **somfTDictionaryInitL** (in long sizeHint);	Initializes a new dictionary, given its initial size.
somf_TDictionary **somfTDictionaryInitLL** (in long sizeHint, in long growthRate);	Initializes a new dictionary, given its initial size and its initial growth rate.
somf_TDictionary **somfTDictionaryInitLLF** (in long sizeHint, in long growthRate, in somf_MCollectibleCompareFn testfn);	Initializes a new dictionary, given its initial size, its initial growth rate, and its comparison test function. Note: This method is equivalent to the somfTDictionaryInitFLL method.
somf_MCollectible **somfValueAt** (in somf_MCollectible key);	Gets the value associated with a given key for a (key, value) pair in a dictionary.

somf_TDictionaryIterator class

(see tdict.idl)

somf_MCollectible **somfFirst** ( );
Resets the iterator and returns the first (key, value) pair from a dictionary.

somf_MCollectible **somfNext** ( );
Gets the next (key, value) pair in the dictionary of a given dictionary iterator.

void **somfRemove** ( );
Removes the current (key, value) pair (the one just returned by somfFirst or somfNext) from the dictionary.

somf_TDictionaryIterator
somfTDictionaryIteratorInit (
in somf_TDictionary h);
Initializes a somf_TDictionaryIterator iterator for a specified dictionary.

somf_THashTable class

(see thash.idl)

somf_MCollectible **somfAddMM** (
in somf_MCollectible key,
in somf_MCollectible value);
Adds a (key, value) pair to the hash table. This method will replace a copy (a pair having the same key) if one already exists.

somf_MCollectible **somfAddMMB** (
in somf_MCollectible key,
in somf_MCollectible value,
in boolean replace);
Adds a (key, value) pair to the hash table, unless the Boolean argument prohibits replacement of a copy (a pair with the same key).

void **somfAssign** (
in somf_THashTable source);
Assigns a hash table as being equal to a given source hash table.

long **somfCount** ( );
Gets the number of objects in the hash table.

somf_MCollectible **somfDelete** (
in somf_MCollectible key);
Deletes a given key and removes the associated (key, value) pair from a hash table, returning a pointer to the value that was removed.

void **somfDeleteAll** ( );
Removes all of the (key, value) pairs from a hash table and deallocates the storage that these pairs might have owned. (That is, the destructor function is called for each object in the hash table).

void **somfDeleteAllKeys** ( );
Removes all of the (key, value) pairs from a hash-table receiving object. However, the program still owns the values of the (key, value) pairs.

void **somfDeleteAllValues** ( );
Removes all of the (key, value) pairs from a hash table. However, the program still owns the keys of the (key, value) pairs.

long **somfGetGrowthRate** ( );
Gets the growth rate of a given hash table.

somf_MCollectibleHashFn
somfGetHashFunction ( );
Gets the hash function used by a given hash table.

long **somfGetRehashThreshold** ( );
Gets the rehash threshold of a given hash table.

void **somfGrow** ( );
Grows a given hash table.

somf_MCollectible **somfMember** (
in somf_MCollectible key);
Gets the (key, value) pair corresponding to a given key for a hash table.

somf_MCollectible **somfRemove** (in somf_MCollectible key);	Removes from the hash table the (key, value) pair associated with a given key.
void **somfRemoveAll** ( );	Removes all of the (key, value) pairs from a hash table.
somf_MCollectible **somfRetrieve** (in somf_MCollectible key);	Retrieves the value associated with a given key for a (key, value) pair in a hash table.
void **somfSetGrowthRate** (in long rate);	Sets the growth rate of a hash table.
void **somfSetHashFunction** (in somf_MCollectibleHashFn fn);	Sets a hash table's hash function to a given function.
void **somfSetRehashThreshold** (in long threshold);	Sets the rehash threshold of a hash table.
somf_THashTable **somfTHashTableInitFL** (in somf_MCollectibleCompareFn testfn, in long tablesize);	Initializes a new hash table, given its comparison test function and its initial table size.
somf_THashTable **somfTHashTableInitFLL** (in somf_MCollectibleCompareFn testfn, in long tablesize, in long rate);	Initializes a new hash table, given its comparison test function, its initial table size, and its initial growth rate.
somf_THashTable **somfTHashTableInitFLLL** (in somf_MCollectibleCompareFn testfn, in long tablesize, in long rate, in long threshold);	Initializes a new hash table, given its comparison test function, its initial table size, its initial growth rate, and its rehash threshold.
somf_THashTable **somfTHashTableInitH** (in somf_THashTable h);	Initializes a new hash table, setting it equal to another specified hash table.

somf_THashTableIterator class

(see thashitr.idl)

somf_MCollectible **somfFirst** ( );	Resets the iterator and returns the first (key, value) pair of a hash table.
somf_MCollectible **somfNext** ( );	Gets the next (key, value) pair from the hash table of a given hash-table iterator.
void **somfRemove** ( );	Removes the current (key, value) pair (the one just returned by somfFirst or somfNext) from the hash table.
somf_THashTableIterator **somfTHashTableIteratorInit** (in somf_THashTable h);	Initializes a somf_THashTableIterator iterator, given its corresponding hash table.

somf_TIterator class

(see titeratr.idl)

somf_MCollectible **somfFirst** ( );	Resets the iterator and returns the first object of a collection.
somf_MCollectible **somfNext** ( );	Gets the next object in a collection.
void **somfRemove** ( );	Removes the current object (the one just returned by somfFirst or somfNext) from a collection.

somf_TPrimitiveLinkedList class

(see tpll.idl)

void **somfAddAfter** (in somf_MLinkable existing, in somf_MLinkable obj);	Adds an object into a list after a given existing object.
void **somfAddBefore** (in somf_MLinkable existing, in somf_MLinkable obj);	Adds an object into a list before a given existing object.
void **somfAddFirst** (in somf_MLinkable obj);	Adds an object as the first object in a list.
void **somfAddLast** (in somf_MLinkable obj);	Adds an object as the last object in a given list.
somf_MLinkable **somfAfter** (in somf_MLinkable existingobj);	Gets the object that comes after a given existing object in a list.
somf_MLinkable **somfBefore** (in somf_MLinkable existingobj);	Returns the object that comes before a given existing object in a list.
unsigned long **somfCount** ( );	Gets the number of objects in a given list.
somf_MLinkable **somfFirst** ( );	Gets the first object in a given list.
somf_MLinkable **somfLast** ( );	Gets the last object in a given list.
void **somfRemove** (in somf_MLinkable aLink);	Removes a somf_MLinkable object from a given list.
void **somfRemoveAll** ( );	Removes all of the objects from a given list.
somf_MLinkable **somfRemoveFirst** ( );	Removes the first object from a given list.
somf_MLinkable **somfRemoveLast** ( );	Removes the last object from a given list.

somf_TPrimitiveLinkedListIterator class

(see tpllitr.idl)

somf_MLinkable **somfFirst** ( );	Resets the iterator and returns the first element of a given list.
somf_MLinkable **somfLast** ( );	Retrieves the last object from a given list.
somf_MLinkable **somfNext** ( );	Gets the next object in a list.
somf_MLinkable **somfPrevious** ( );	Gets the previous object from a given list.
somf_TPrimitiveLinkedListIterator **somfTPrimitiveLinkedListIteratorInit** (in somf_TPrimitiveLinkedList list);	Initializes a somf_TPrimitiveLinkedListIterator object, establishing it as the iterator for a given somf_TPrimitiveLinkedList linked list.

somf_TPriorityQueue class

(see tpq.idl)

somf_MCollectible **somfAdd** (in somf_MCollectible obj);	Adds a given obj to a priority queue.
void **somfAssign** (in somf_TPriorityQueue source);	Assigns the priority-queue receiving object as being equal to a given source priority queue.

long **somfCount** ( );	Gets the number of objects in a given priority queue.
somf_TIterator **somfCreateIterator** ( );	Returns a new iterator that is suitable for iterating over the objects in a given priority queue.
void **somfDeleteAll** ( );	Removes all of the objects from the priority-queue receiving object and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.)
somf_MCollectibleCompareFn **somfGetEqualityComparisonFunction** ( );	Gets the equality comparison function being used by the priority queue. The default equality compare function is the somf_MCollectible class's somfIsEqual**.**
void **somfInsert** (in somf_MOrderableCollectible obj);	Inserts an object obj into the priority queue.
somf_MCollectible **somfMember** (in somf_MCollectible obj);	Gets an object from a given priority queue.
somf_MOrderableCollectible **somfPeek** ( );	Determines the object with the "highest" priority in the priority queue, but does not remove it.
somf_MOrderableCollectible **somfPop** ( );	Gets the object with the "highest" priority from a given priority queue.
somf_MCollectible **somfRemove** (in somf_MCollectible obj);	Removes an object obj from a given priority queue.
void **somfRemoveAll** ( );	Removes all of the objects from a given priority queue.
somf_MOrderableCollectible **somfReplace** (in somf_MOrderableCollectible obj);	Removes the object with the highest priority from a given priority queue, and then inserts an object obj into the priority queue.
void **somfSetEqualityComparisonFunction** (in somf_MCollectibleCompareFn fn)	Sets a member function to be called as the equality comparison function when removing objects from the queue, checking whether a given object is a member, and so on.
somf_TPriorityQueue **somfTPriorityQueueInitF** (in somf_MOrderableCompareFn testfn);	Initializes a new priority queue, given a comparison test function.
somf_TPriorityQueue **somfTPriorityQueueInitP** (in somf_TPriorityQueue q);	Initializes a new priority queue, setting it equal to another specified priority queue.

somf_TPriorityQueueIterator class

(see tpqitr.idl)

somf_MCollectible **somfFirst** ( );	Resets the iterator and returns the first object in a priority queue.
somf_MCollectible **somfNext** ( );	Gets the next object in a priority queue.
void **somfRemove** ( );	Removes the current object (the one just returned by a somfFirst or somfNext method) from a priority queue.

somf_TPriorityQueueIterator **somfTPriorityQueueIteratorInit** (in somf_TPriorityQueue h);	Initializes a new priority queue iterator, given the priority queue over which it will iterate.

somf_TSequence class

(see tseq.idl)

somf_MCollectible **somfAdd** (in somf_MCollectible obj);	Adds an object to a given ordered collection.
somf_MCollectible **somfAfter** (in somf_MCollectible obj);	Gets the object found after a given object obj in an ordered collection.
somf_MCollectible **somfBefore** (in somf_MCollectible obj);	Gets the object found before a given obj in an ordered collection.
long **somfCount** ( );	Gets the number of objects in this ordered collection.
somf_TIterator **somfCreateIterator** ( );	Returns a new iterator that is suitable for iterating over the objects in an ordered collection.
void **somfDeleteAll** ( );	Removes all of the objects from an ordered collection and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.)
somf_MCollectible **somfFirst** ( );	Gets the first object in an ordered collection.
somf_MCollectible **somfLast** ( );	Gets the last object in an ordered collection.
long **somfOccurrencesOf** (in somf_MCollectible obj);	Determines the number of times an object obj is contained in an ordered collection.
somf_MCollectible **somfRemove** (in somf_MCollectible obj);	Removes an object from an ordered collection.
void **somfRemoveAll** ( );	Removes all of the objects from an ordered collection.
somf_TSequence **somfTSequenceInit** (in somf_MCollectibleCompareFn testfn);	Initializes a new ordered collection of class somf_TSequence, given a comparison function for the collection to use.

somf_TSequenceIterator class

(see tseqitr.idl)

somf_MCollectible **somfFirst** ( );	Resets the iterator and gets the first object of an ordered collection.
somf_MCollectible **somfLast** ( );	Gets the last object in an ordered collection.
somf_MCollectible **somfNext** ( );	Gets the next object in an ordered collection.
somf_MCollectible **somfPrevious** ( );	Gets the previous object in an ordered collection.
void **somfRemove** ( );	Removes the current object from ordered collection.

somf_TSet class

(see tset.idl)

somf_MCollectible **somfAdd** (in somf_MCollectible obj);	Adds an object to a given set.
void **somfAssign** (in somf_TSet source);	Assigns a set as being equal to a given source set.
long **somfCount** ( );	Gets the number of objects in a given set.
somf_TIterator **somfCreateIterator** ( );	Returns a new iterator that is suitable for iterating over the objects in this set.
void **somfDeleteAll** ( );	Removes all of the objects from a set and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the set.)
void **somfDifferenceS** (in somf_TSet set1);	Determines the elements of a given set that do not appear in another specified set, and modifies the first set to contain only those different elements.
void **somfDifferenceSS** (in somf_TSet set1, in somf_TSet resultSet);	Determines the elements of a given set that do not appear in another specified set, and places those different elements in a third set.
somf_MCollectibleHashFn **somfGetHashFunction** ( );	Gets a pointer to the hash function used by a set.
void **somfIntersectionS** (in somf_TSet set1);	Gets those elements that are members of both a given set and another set, set1, and modifies the first set to contain only those common elements.
void **somfIntersectionSS** (in somf_TSet set1, in somf_TSet resultSet);	Gets those elements that are members of both a given set and another set, set1, and places those common elements in a third set.
somf_MCollectible **somfMember** (in somf_MCollectible obj);	Gets an object from a set.
void **somfRehash** ( );	Rehashes a set, cleaning up for any objects that were marked for deletion.
somf_MCollectible **somfRemove** (in somf_MCollectible obj);	Removes an object from a given set.
void **somfRemoveAll** ( );	Removes all of the objects from a given set.
void **somfSetHashFunction** (in somf_MCollectibleHashFn fn);	Sets the hash function of a set.
somf_TSet **somfTSetInitF** (in somf_MCollectibleCompareFn testfn);	Initializes a new set, given its comparison test function.
somf_TSet **somfTSetInitFL** (in somf_MCollectibleCompareFn testfn, in long setSizeHint);	Initializes a new set, given the comparison test function and the initial set size. Note: This method is equivalent to the method somfTSetInitLF**.**
somf_TSet **somfTSetInitL** (in long setSizeHint);	Initializes a new set, given the initial set size.

somf_TSet **somfTSetInitLF** (in long setSizeHint, in somf_MCollectibleCompareFn testfn);	Initializes a new set, given the initial set size and the comparison test function. Note: This method is equivalent to method somfTSetInitFL**.**
somf_TSet **somfTSetInitS** (in somf_TSet s);	Initializes a new set, establishing it as equal to another given set.
void **somfUnionS** (in somf_TSet set1);	Gets those elements that are members of either a given set or another set, set1, and modifies the first set to contain all elements from either set.
void **somfUnionSS** (in somf_TSet set1, in somf_TSet resultSet);	Gets those elements that are members of either a given set and another set, set1, and places all those elements in a third set.
void **somfXorS** (in somf_TSet set1);	Determines a set wherein each member is an element of either a given set or another set set1*,* but not both, and modifies the first set to contain the elements of the new set.
void **somfXorSS** (in somf_TSet set1, in somf_TSet resultSet);	Determines a set where each member is an element either of a given set or of another set set1*,* but not both, and places all of those elements in a third set.

somf_TSetIterator class

(see tsetitr.idl)

somf_MCollectible **somfFirst** ( );	Resets the iterator and returns the first element of a set.
somf_MCollectible **somfNext** ( );	Gets the next object in a set.
void **somfRemove** ( );	Removes the current object (the one just returned by somfFirst or somfNext) from a set.
somf_TSetIterator **somfTSetIteratorInit** (in somf_TSet h);	Initializes somf_TSetIterator object, establishing it as the iterator for a given somf_TSet set.

somf_TSortedSequence class

(see tss.idl)

somf_MCollectible **somfAdd** (in somf_MCollectible obj);	Adds an obj to a sorted sequence.
somf_MCollectible **somfAfter** (in somf_MCollectible obj);	Gets the object found after a given object in a sorted sequence.
void **somfAssign** (in somf_TSortedSequence s);	Assigns a sorted sequence as equal to a given source sorted sequence.
somf_MCollectible **somfBefore** (in somf_MCollectible obj);	Returns the object found before a given object in a sorted sequence.
long **somfCount** ( );	Gets the number of objects in a given sorted sequence.
somf_TIterator **somfCreateIterator** ( );	Returns a new iterator that is suitable for iterating over the objects in a sorted sequence.

somf_TSequenceIterator
somfCreateSequenceIterator ( );

Returns a new iterator that is suitable for iterating over the objects in a sorted sequence.

somf_TSortedSequenceNode
somfCreateSortedSequenceNode (
in somf_TSortedSequenceNode n1,
in somf_MOrderableCollectible obj,
in somf_TSortedSequenceNode n2);

Creates a new somf_TSortedSequenceNode in a somf_TSortedSequence collection, given a key to the new node and its left and right children.

void **somfDeleteAll** ( );

Removes all objects from a sorted sequence and deallocates the storage that these objects might have owned. (That is, the destructor function is called for each object in the collection.)

somf_MCollectible **somfFirst** ( );

Gets the first object in a sorted sequence.

somf_MBetterOrderableCompareFn
somfGetSequencingFunction ( );

Gets a pointer to the function used to compare objects in a sorted sequence, and consequently determines the sequence of the collection.

somf_MCollectible **somfLast** ( );

Gets the last object in a sorted sequence.

somf_MCollectible **somfMember** (
in somf_MCollectible obj);

Gets an object in a sorted sequence.

long **somfOccurrencesOf** (
in somf_MCollectible obj);

Determines the number of times an object is in a sorted sequence.

somf_MCollectible **somfRemove** (
in somf_MCollectible obj);

Removes an object from a sorted sequence.

void **somfRemoveAll** ( );

Removes all of the objects from a sorted sequence.

void **somfSetSequencingFunction** (
in somf_MBetterOrderableCompareFn fn);

Sets a pointer to the function used to compare objects in a sorted sequence, and consequently determines the sequence of the collection.

somf_TSortedSequence
somfTSortedSequenceInitF (
in somf_MBetterOrderableCompareFn testfn);

Initializes a new sorted sequence, given the comparison method that it will use.

somf_TSortedSequence
somfTSortedSequenceInitS (
in somf_TSortedSequence s);

Initializes a new sorted sequence, setting it equal to another given sorted sequence.

somf_TSortedSequenceIterator class

(see tssitr.idl)

somf_MCollectible **somfFirst** ( );

Resets the iterator and returns the first object of a sorted sequence.

somf_MCollectible **somfLast** ( );

Gets the last object in a sorted sequence.

somf_MCollectible **somfNext** ( );

Gets the next object in a sorted sequence.

somf_MCollectible **somfPrevious** ( );

Gets the previous object in a sorted sequence.

void **somfRemove** ( );

Removes the current object (the one just returned by a somfFirst or somfNext method) from a sorted sequence.

somf_MOrderableCollectible
somfStartHere (
in somf_MOrderableCollectible obj);

Starts iteration of a somf_TSortedSequence, beginning at a given object obj, rather than at the front of the collection.

somf_TSortedSequenceIterator
somfTSortedSequenceIteratorInit (
in somf_TSortedSequence h);

Initializes a new somf_TSortedSequenceIterator object, given the somf_TSortedSequence collection over which it will iterate.

somf_TSortedSequenceNode

(see tssnode.idl)

somf_MOrderableCollectible **somfGetKey** ( );

Gets the key to a node.

somf_TSortedSequenceNode
somfGetLeftChild ( );

Gets the left child of a node.

somf_TSortedSequenceNode
somfGetParent ( );

Gets the parent of a node.

boolean **somfGetRed** ( );

Determines whether a node is red or black.

somf_TSortedSequenceNode
somfGetRightChild ( );

Gets the right child of a node.

void **somfSetKey** (
in somf_MOrderableCollectible k);

Sets the key to a node, given a pointer to a key object.

void **somfSetLeftChild** (
in somf_TSortedSequenceNode n);

Sets the left child of a node, given a pointer to an object that will be the left child.

void **somfSetParent** (
in somf_TSortedSequenceNode n);

Sets the parent of a node, given an object to be used as the parent node.

void **somfSetRed** (in boolean on);

Sets a node to red or black.

void **somfSetRedOn** ( );

Sets a node to red.

void **somfSetRightChild** (
in somf_TSortedSequenceNode n);

Sets the right child of a node, given a pointer to an object that will be the right child.

somf_TSortedSequenceNode
somfTSortedSequenceNodeInitT (
in somf_TSortedSequenceNode n1);

Initializes a new somf_TSortedSequenceNode node, given its left child.

somf_TSortedSequenceNode
somfTSortedSequenceNodeInitTM (
in somf_TSortedSequenceNode n1,
in somf_MOrderableCollectible obj);

Initializes a new somf_TSortedSequenceNode node, given its left child and a key to the new node.

somf_TSortedSequenceNode
somfTSortedSequenceNodeInitTMT (
in somf_TSortedSequenceNode n1,
in somf_MOrderableCollectible obj,
in somf_TSortedSequenceNode n2);

Initializes a new somf_TSortedSequenceNode node, given a key to the new node and its left and right children.

Index

T

U

W