# Bull DPX/20

## SOMobject Base Toolkit
## Quick Reference Guide

AIX

ORDER REFERENCE
**86 A2 29AQ 01**

# Bull DPX/20

## SOMobject Base Toolkit
## Quick Reference Guide

AIX

**Software**

**June 1995**

**BULL S.A. CEDOC**
Atelier de Reproduction
FRAN–231
331 Avenue Patton BP 428
49005 ANGERS CEDEX
FRANCE

ORDER REFERENCE
**86 A2 29AQ 01**

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

## Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

AIX® is a registered trademark of International Business Machines Corporation, and is being used under licence.

UNIX is a registered trademark in the USA and other countries licensed exclusively through X/Open.

# About This Book

This book is a handy reference, much like a “pocket reference guide.” It gives the syntax and a one-sentence purpose statement for every method, function, and macro defined in the **SOMobjects Base Toolkit**, Version 2.0, plus a listing of the SOM Compiler commands/flags. The references are organized by each framework of the **SOMobjects Base Toolkit** and are further subdivided by class. The *Quick Reference Guide* is abstracted from the *SOMobjects Base Toolkit Programmer’s Reference Manual*. Experienced programmers will find the *Quick Reference* a convenient reminder of keywords and parameters, without having to refer to the complete *Programmer’s Reference Manual*.

For complete explanations of the purpose, syntax parameters, return type, and an example of the SOMobjects methods, functions, and macros, refer to the *SOMobjects Base Toolkit Programmer’s Reference Manual*. In addition, refer to the *SOMobjects Base Toolkit Users Guide* for introductory information.

# Who Should Use This Book

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

This book assumes that you are an experienced programmer and that you have a general familiarity with the basic notions of object-oriented programming. In addition, it assumes that you are already familiar with the SOMobjects system and with the information presented in the related manuals for the **SOMobjects Base Toolkit.**

# 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**

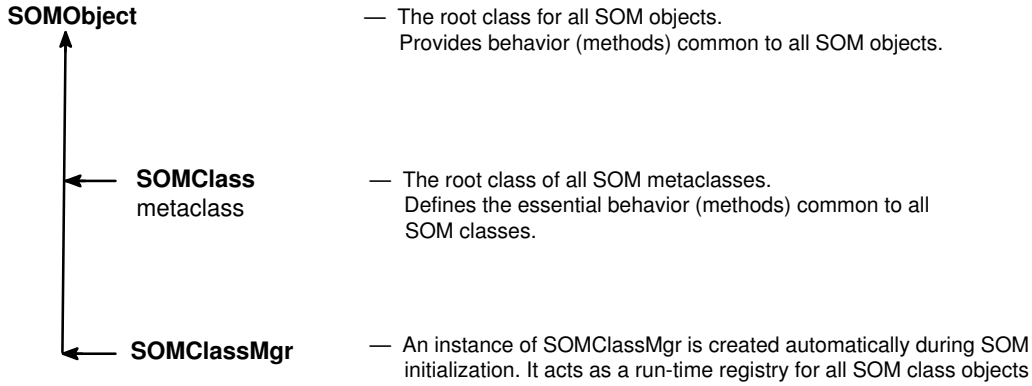
Install public versions of the .idl files in the indicated directory.

**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 **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);

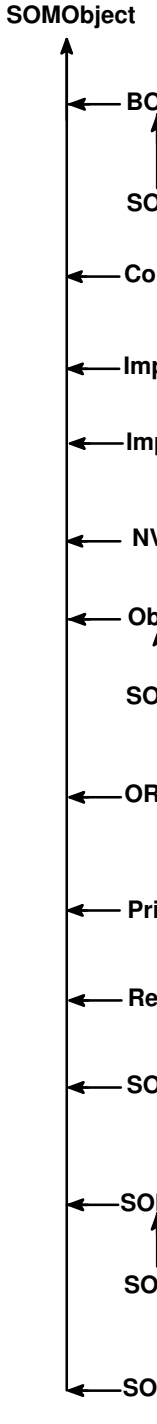
Tests whether an object can respond to an invocation of the indicated method.

void **somUninit** ();

Un-initializes 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 );
```

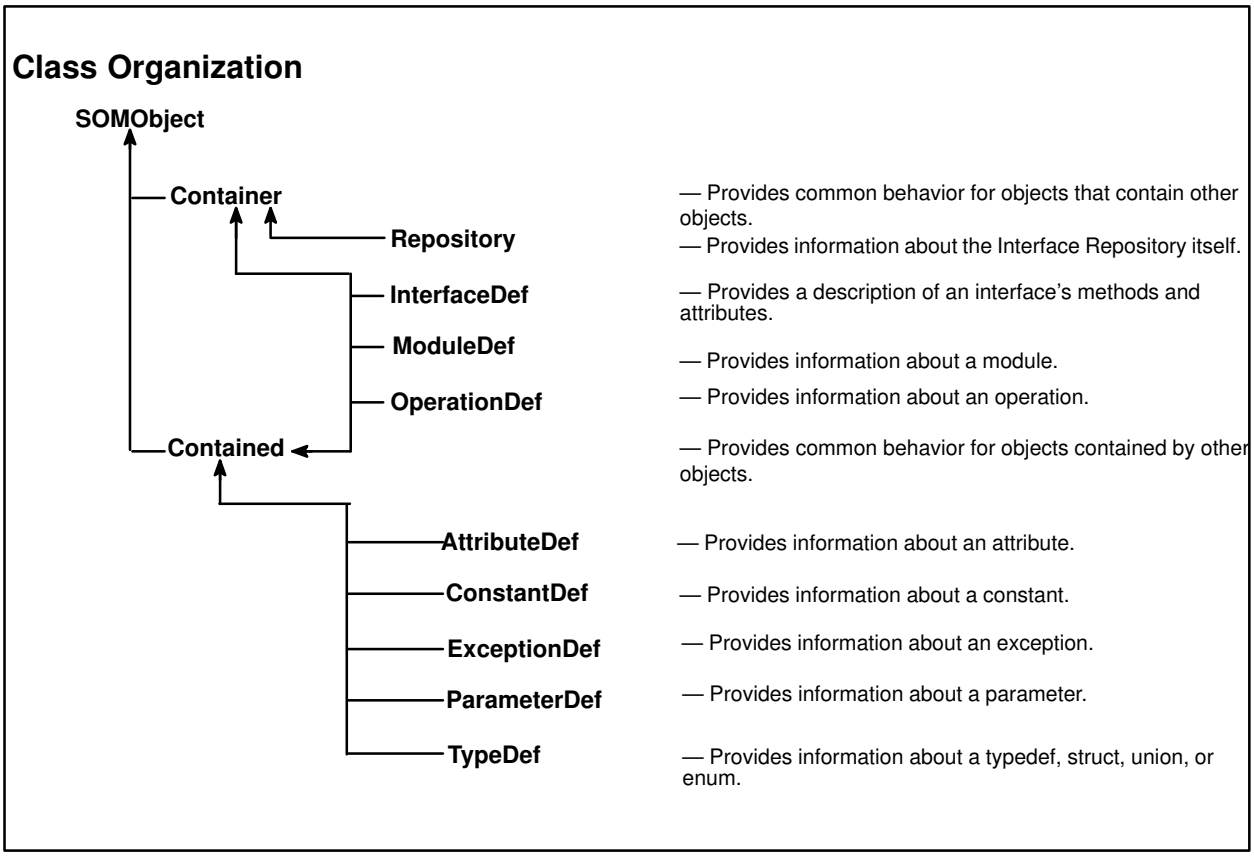
Continuously receives request messages, executing them and returning results.

```
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.

## Interface Repository 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);

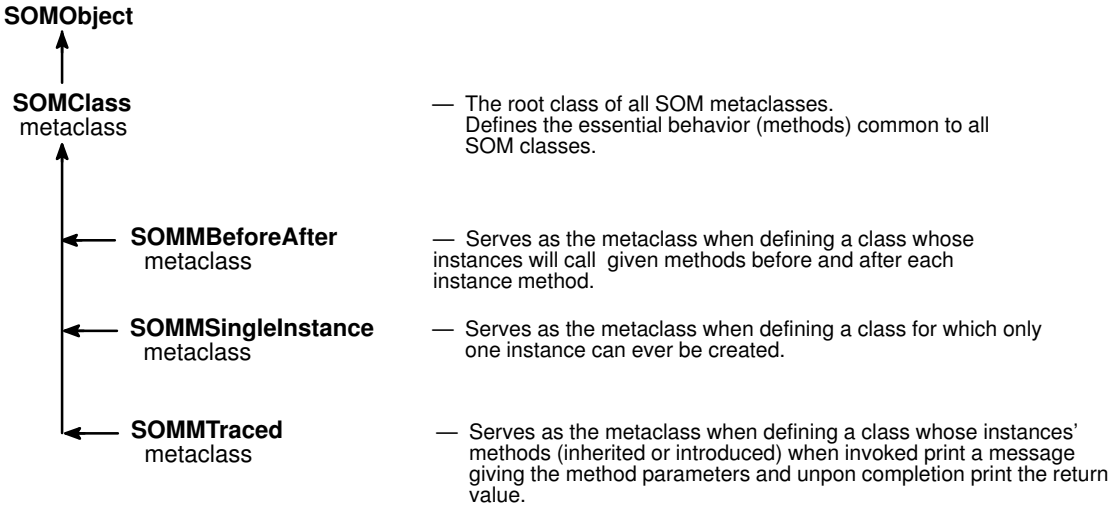
Overrides the default alignment associated with the given TypeCode.

long **TypeCode_size** ( );

Provides the minimum size of an instance of the abstract data type described by a given TypeCode.

# 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

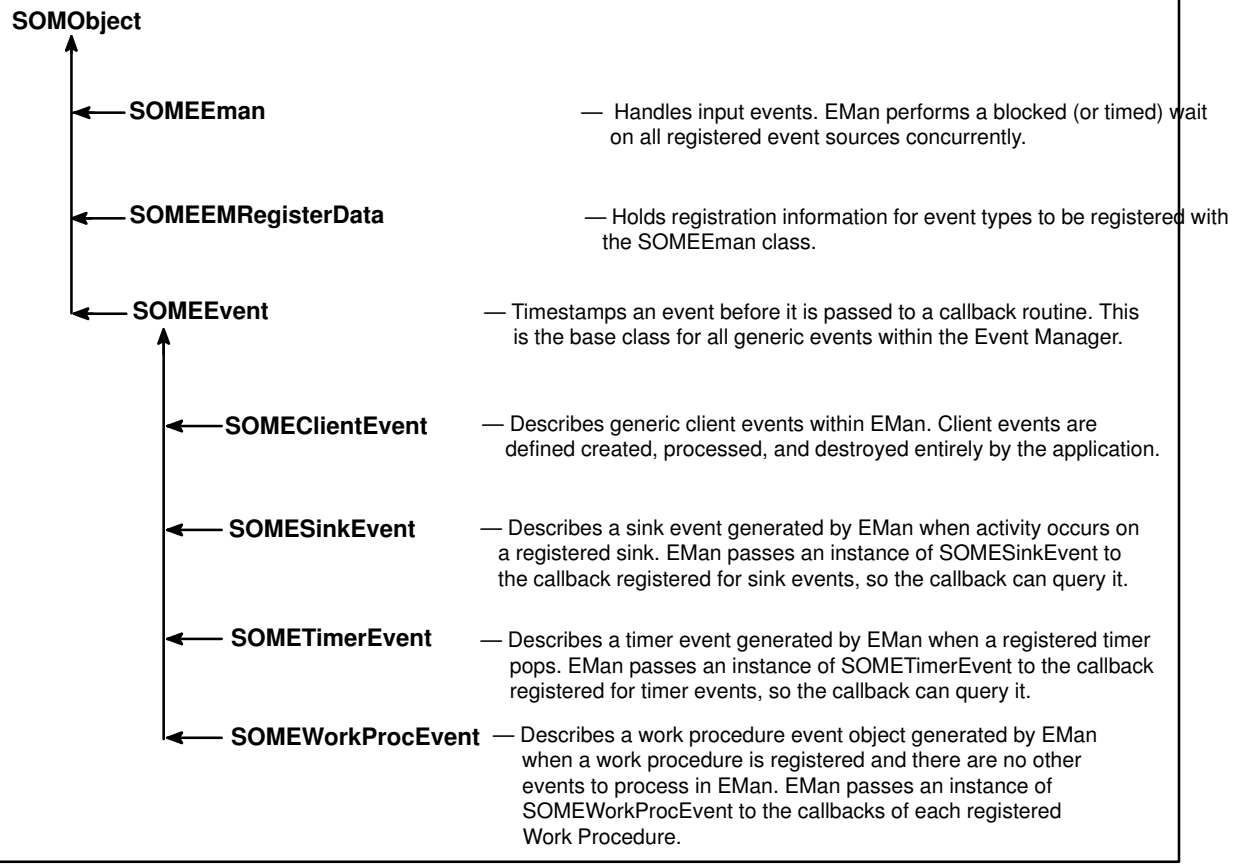
(see somtrcls.idl)

attribute boolean **sommTraceIsOn**

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

# 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).

# Index

# P



# Q



# R



# S

## T

# U



# W

# Vos remarques sur ce document / Technical publication remark form

**Titre / Title :** Bull DPX/20 SOMobject Base Toolkit Quick Reference Guide

**Nº Reférence / Reference Nº :** 86 A2 29AQ 01

**Daté / Dated :** June 1995

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : _____ Date : _____

SOCIETE / COMPANY : _____

ADRESSE / ADDRESS : _____

_____

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

**BULL S.A. CEDOC**
Atelier de Reproduction
FRAN–231
331 Avenue Patton BP 428
49005 ANGERS CEDEX
FRANCE

PLACE BAR CODE IN LOWER LEFT CORNER

**BULL S.A. CEDOC**
Atelier de Reproduction
FRAN–231
331 Avenue Patton BP 428
49005 ANGERS CEDEX
FRANCE

ORDER REFERENCE
**86 A2 29AQ 01**

Bull

Utiliser les marques de découpe pour obtenir les étiquettes.
Use the cut marks to get the labels.

**DPX/20**
AIX
SOMobject Base Toolkit Quick Reference Guide
86 A2 29AQ 01

**DPX/20**
AIX
SOMobject Base Toolkit Quick Reference Guide
86 A2 29AQ 01

**DPX/20**
AIX
SOMobject Base Toolkit Quick Reference Guide
86 A2 29AQ 01