OS/390

SOMobjects
Messages, Codes and Diagnosis

SC28-1996-01

OS/390

SOMobjects Messages, Codes and Diagnosis

SC28-1996-01

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vi.

Second Edition, September 1997

| This edition applies to Version 2 Release 4 of OS/390 (5647-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):
Your International Access Code +1+914+432-9405

IBMLink (United States customers only): KGNVMC(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet: mhvrcfs@vnet.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Notices

Examples in This Book

Programming Interface Information

This publication is intended to help you diagnose and debug SOMobjects messages, codes and related problems and primarily documents Diagnosis, Modification, or Tuning Information.

Attention: Do not use this Diagnosis, Modification or Tuning Information as a programming interface.

However, this publication also documents General-use Programming Interface and Associated Guidance Information provided by SOMobjects.

General-use programming interfaces allow the customer to write programs that obtain the services of SOMobjects.

Both Diagnosis, Modification, or Tuning Information and General-use Programming Interface and Associated Guidance Information are identified where they occur by an introductory statement to a chapter or section.

Trademarks

The following terms are trademarks for the IBM Corporation in the United States and/or other countries:

AD/Cycle
AIX
AIX/6000
C++/MVS
C/MVS
C/370
CICS
Common User Access
DB2
IBM
IBMLink
IMS
MVS/ESA
OpenEdition
OS/2
Presentation Manager
PS/2
RACF
RMF
RS/6000
SystemView
S/390

The following terms, **DENOTED BY A DOUBLE ASTERISK (**)**, used in this publication, are trademarks of other companies as follows:

UNIX	X/Open Company Limited
Windows	Microsoft Corporation

Windows is a trademark of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

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

The term “ANSI C” used throughout this publication refers to American National Standard X3.159-1989.

About This Book

This book includes information on diagnosing or debugging SOMobjects problems.

Who Should Use This Book

| This book is for the professional programmer using C, C++, or IBM COBOL for
| OS/390 and VM Version 2 Release 1 who wishes to diagnose and debug problems with SOMobjects using the trace facilities, error logs, messages and codes that are included in this publication.

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

This book uses the common terminology of object-oriented programming. A number of important terms are everyday English words that take on specialized meanings. These terms appear in the Glossary at the back of this book. You might consult the Glossary if the unusual significance attached to an otherwise ordinary word puzzles you.

How This Book is Organized

This book is divided into seven chapters and an appendix:

- Chapter 1, "Generating Diagnostic Messages From Your Program" on page 1-1

  The audience for this chapter is the SOMobjects application developer and user who wants to prepare their application for debugging. This chapter consists of the following topics:

  – "Using Macro Interfaces" on page 1-2
  – "Controlling Debug Output" on page 1-1
  – "Using Class and Method Interfaces" on page 1-3
  – "Using Replaceable Routines" on page 1-4

- Chapter 2, "Handling Exceptions" on page 2-1 which includes:

  – "Introduction to Exceptions" on page 2-1
  – "The Environment" on page 2-3
  – "Setting an Exception Value" on page 2-3
  – "Getting an Exception Value" on page 2-4
  – "Example Of Raising an Exception" on page 2-5

- Chapter 3, "Using SOM Trace Facilities" on page 3-1

  This section describes how to debug problems with SOM trace facilities and consists of the following topics:

  – "Capturing Trace Records" on page 3-1
  – "Displaying Captured Trace Records" on page 3-2
  – "Combining the Output of Multiple Traces" on page 3-3
  – "Displaying SOM Trace Data from an OS/390 Dump" on page 3-3
  – "Sample JCL to Display SOM Trace Data" on page 3-4

  - "Writing Applications to Use SOM Trace Facilities" on page 3-4
- Chapter 4, "Using the Error Log Facility" on page 4-1 consists of the following topics:
  - "Configuring the Error Log" on page 4-1
  - "Using The Error Log" on page 4-2
  - "Understanding Error Log Entries" on page 4-3
  - "Locating the Correct Log File" on page 4-5
- Chapter 5, "SOMobjects Messages" on page 5-1, provides the messages you can receive with the SOMobjects frameworks.
- Chapter 6, "6C4 System Abend" on page 6-1, provides a description of the 6C4 SOMobjects system abend that can occur on OS/390, with its error codes and associated explanations and responses.
- Chapter 7, "SOMobjects Error Codes" on page 7-1, provides the error codes you can receive with the SOMobjects frameworks.
- Appendix A, "Troubleshooting Hints" on page A-1 is divided into four sections:
  - Chapter 6, "6C4 System Abend" on page 6-1 is the description of the 6C4 OS/390 SOMobjects system abend with its error codes and associated explanations and responses.
  - "Interface Repository Dump (IRDUMP) Utility" on page A-1
  - "Checking the DSOM Setup" on page A-2
  - "Analyzing Problem Symptoms" on page A-2
  - "Unexplained Program Crashes" on page A-4
  - "Reporting Problems to IBM" on page A-5

Where to Find More Information

The book references the following publications using the shortened version of the book title. The following table lists the shortened titles, complete titles, and order numbers of the books you might need while you are using this book.

Table 0-1 (Page 1 of 2). SOMobjects Publications

Short Title Used in This Book	Title	Order Number
OS/390 V2R4.0 SOMobjects: Getting Started	*OS/390 SOMobjects: Getting Started*	GA22-7248
OS/390 V2R4.0 SOMobjects Messages, Codes, and Diagnosis	*OS/390 SOMobjects Messages, Codes, and Diagnosis*	SC28-1996
OS/390 V2R4.0 SOMobjects Object Services	*OS/390 SOMobjects Object Services*	SC28-1995
OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1	*OS/390 SOMobjects Programmer's Reference, Volume 1*	SC28-1997
OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 2	*OS/390 SOMobjects Programmer's Reference, Volume 2*	SC28-1998
OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 3	*OS/390 SOMobjects Programmer's Reference, Volume 3*	SC28-1999
OS/390 V2R4.0 SOMobjects Configuration and Administration Guide	*OS/390 SOMobjects Configuration and Administration Guide*	GC28-1851

Table 0-1 (Page 2 of 2). SOMobjects Publications

Short Title Used in This Book	Title	Order Number
OS/390 V2R4.0 SOMobjects Programmer's Guide	*OS/390 SOMobjects Programmer's Guide*	GC28-1859

SOMobjects books are also available in softcopy on the OS/390 Collection (SK2T-6700) CD-ROM. IBM provides one copy of the CD-ROM automatically with the basic material for OS/390. You can order additional copies for a fee.

CORBA Publications

- *The Common Object Request Broker: Architecture and Specification*, published by the Object Management Group and X/Open.

C++ Publications

- *Algorithms in C++* by Robert Sedgewick (Addison-Wesley Publishing Company, 1992)

International Technical Support Organization Publications

For additional information on SOMobjects, see the following publications:

Table 0-2. International Technical Support Organization Publications

Title	Order Number
Object Technology in Application Development	GG24-4290
A Survey of Object-Oriented Technology for MVS	GG24-2505

Chapter 1. Generating Diagnostic Messages From Your Program

The audience for this chapter is the SOMobjects application developer and user who wants to prepare their application for debugging. This chapter consists of the following topics:

- "Controlling Debug Output"
- "Using Macro Interfaces" on page 1-2
- "Using Class and Method Interfaces" on page 1-3
- "Using Replaceable Routines" on page 1-4

This chapter documents General-use Programming Interface and Associated Guidance Information provided by SOMobjects.

Controlling Debug Output

Debugging output is produced or suppressed based on the settings of three global variables, **SOM_TraceLevel**, **SOM_WarnLevel** and **SOM_AssertLevel**:

Note: These global variables affect the generation of diagnostic messages from SOMobjects itself, as well as from the client program.

- **SOM_TraceLevel** controls the behavior of the *className***MethodDebug** macro and the behavior of the MVSTrace facility.

  For more information about using SOM_TraceLevel to capture records, see "Capturing Trace Records" on page 3-1.
- **SOM_WarnLevel** controls the behavior of the **SOM_WarnMsg** macro, **SOM_TestC** macro, **SOM_Expect** macro, and **SOM_Assert** macro.
- **SOM_AssertLevel** controls the behavior of the **SOM_Assert** macro, **SOM_TestC** macro, and **SOM_Test** macro.

Checking the Validity of Method Calls

The C and C++ language bindings include code to check the validity of method calls at run time. If a validity check fails, the **SOM_Error** macro ends the process. To enable method-call validity checking, place the following directive in the client program prior to any #include directives for SOM header files:

```
#define SOM_TestOn
```

Alternatively, using

```
DEF(SOM_TestOn)
```

can be used when compiling the client program to enable method-call validity checking.

The **SOMM_TRACED** OS/390 OpenEdition environment variable can be used to control output generated by the **SOMMTraced** metaclass. See "SOMMTraced Metaclass" on page 1-4 for more information.

Using Macro Interfaces

The macros defined below are used to conditionally generate output for debugging.

Output from the following macros are controlled by the **SOM_AssertLevel** and **SOM_WarnLevel** global variables, which are described in “Controlling Debug Output” on page 1-1.

SOM_TestC macro

The **SOM_TestC** macro takes as an argument a boolean expression. If the boolean expression is TRUE (nonzero) and **SOM_AssertLevel** is greater than zero, then an informational message is output. If the expression is FALSE (zero) and **SOM_WarnLevel** is greater than zero, a warning message is produced.

SOM_WarnMsg macro

The **SOM_WarnMsg** macro, takes as an argument a character string. If the value of **SOM_WarnLevel** is greater than zero, the specified message is output.

SOM_Assert macro

The **SOM_Assert** macro takes as arguments a boolean expression and an error code (an integer). If the boolean expression is TRUE (nonzero) and **SOM_AssertLevel** is greater than zero, then an informational message is output. If the expression is FALSE (zero), and the error code indicates a warning-level error and **SOM_WarnLevel** is greater than zero, then a warning message is output. If the expression is FALSE and the error code indicates a fatal error, then an error message is produced and the process is terminated.

SOM_Expect macro

The **SOM_Expect** macro takes as an argument a boolean expression. If the boolean expression is FALSE (zero) and **SOM_WarnLevel** is set to be greater than zero, then a warning message is output. If the expression is TRUE and **SOM_AssertLevel** is set to be greater than zero, then an informational message is output.

SOM_Test macro

The **SOM_Test** macro takes a boolean expression as an argument. If the expression is TRUE (nonzero) and the **SOM_AssertLevel** is greater than zero, then an informational message is output. If the expression is FALSE (zero), an error message is produced and the program is terminated.

When errors are encountered in client programs or user defined-classes, the following two macros can be used to invoke the error-handling procedure:

SOM_Error macro

The **SOM_Error** macro takes an error code as its only argument and invokes the SOM error handling procedure (pointed to by the global variable **SOMError**) to handle the error.

SOMError is a replaceable routine. Replacing **SOMError** is described in “Replacing the SOMError Routine” on page 1-4.

The default error handling procedure prints a message that includes the error code, the name of the source file, and the line number where the macro was invoked. If the last digit of the error code indicates a serious error (of category **SOM_Fatal**), the process causing the error is terminated. For more information on **SOM_Fatal**, see “Replacing the SOMError Routine” on page 1-4.

The macro for generating debugging output for the **SOM_TraceLevel** external control is:

*classname***MethodDebug**

*classname***MethodDebug** is the macro for C and C++ programmers using the SOM language bindings for *classname*. The arguments to this macro are a class name and a method name. If the **SOM_TraceLevel** global variable has a nonzero value, the *classname***MethodDebug** macro produces a message each time the specified method (as defined by the specified class) is executed. This macro is typically used within the procedure that implements the specified method. (The SOM Compiler automatically generates calls to the *className***MethodDebug** macro within the implementation template files it produces.) To suppress method tracing for all methods of a class, put the following statement in the implementation file after including the header file for the class:

```
#define classnameMethodDebug(c,m)\
        SOM_NoTrace(c,m)
```

This can yield a slight performance improvement.

The SOM MVSTrace facility provides some tracing features which are an add-on to the existing *classname***MethodDebug** macro. If in your C program you include the mvstrace header prior to your other header includes, you can override the *classname***MethodDebug** with a call to MVSTrace. In that way you can get trace records captured into a trace log. For more information on tracing, see Chapter 3, “Using SOM Trace Facilities” on page 3-1.

The codes for errors detected by SOM are listed in Chapter 7, “SOMobjects Error Codes” on page 7-1. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information on a specific macro.

Using Class and Method Interfaces

This section consists of the following topics:

- SOMMTraced metaclass
- somDumpSelf and somPrintSelf

SOMMTraced Metaclass

The **SOMMTraced** metaclass provides an additional tracing facility that includes method parameters and returned values.

OS/390 SOMobjects also supplies a special metaclass, **SOMMTraced**, that facilitates tracing of method invocations. If a class is an instance of **SOMMTraced** (if SOMMTraced is its metaclass), any method invoked on an instance of that class is traced. That is, before the method begins execution, a message prints (to standard output) giving actual parameters; then, after the method completes execution, a second message prints giving the returned value. To have methods of a particular class traced, simply make **SOMMTraced** its metaclass.

Before using **SOMMTraced**, the OS/390 OpenEdition environment variable “SOMM_Traced” must be set, or no output will be generated. If it is set to an empty string (indicated via use of two double-quotes), all traced classes print. If the environment variable is set to list of names of classes, only those classes will be traced. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information.

somDumpSelf and somPrintSelf

The **somDumpSelf** and **somPrintSelf** methods can be useful in testing and debugging. **somPrintSelf** produces a brief description of an object, and **somDumpSelf** produces a more detailed description. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information.

Using Replaceable Routines

Output from the **SOM_TestC**, **SOM_WarnMsg** macro, **SOM_Assert**, **SOM_Test**, and **SOM_Expect** macro macros is written using the replaceable character-output procedure pointed to by the global variable **SOMOutCharRoutine**. See “Using Macro Interfaces” on page 1-2 for more information on these macros. The default procedure simply writes the character to `stdout`, but it can be replaced to change the output destination of the methods and functions described in “Controlling Debug Output” on page 1-1.

Replacing the SOMError Routine

In the classes provided in the SOM run-time library (that is, **SOMClass**, **SOMObject** and **SOMClassMgr**) error handling is performed by a user-replaceable procedure, pointed to by the global variable **SOMError**, that produces an error message and an error code and, if appropriate, ends the process where the error occurred.

The **SOMError** routine is given control by calling the **SOM_Error** macro. See “Using Macro Interfaces” on page 1-2 for more information on the **SOM_Error** macro.

To replace the **SOMError** routine, store the address of your routine in the **SOMError** global variable. Calling the original **SOMError** routine from your replacement error routine is optional. The prototype for the **SOMError** is:

```
void SOMError (int errCode);
```

Each error is assigned a unique integer parameter. Errors are grouped into three categories, based on the last digit of the error code:

SOM_Ignore This category of error represents an informational event. The event is considered normal and can be ignored or logged at the user's discretion. Error codes ending in a digit 2 belong to this category.

SOM_Warn This category of error represents an unusual condition that is not a normal event, but is not severe enough to require program termination. Error codes ending in a digit 1 belong to this category.

SOM_Fatal This category of error represents a condition that should not occur or that would result in loss of system integrity if processing were allowed to continue. In the default error handling procedure, these errors cause the termination of the process in which they occur. Error codes ending in a digit 9 belong to this category.

The codes for errors detected by SOM are listed in Chapter 7, “SOMobjects Error Codes” on page 7-1.

Chapter 2. Handling Exceptions

Most SOMobjects classes, with the notable exclusion of SOMClass, SOMObject, and SOMClassMgr, return error information in an inout parameter of type **Environment***. Whenever invoking a method that has a parameter of this type, you should immediately test for an exception upon return from the method.

Errors Reported by SOMClass, SOMObject, and SOMClassMgr:

The classes, SOMClass, SOMObject, and SOMClassMgr, report errors by calling the routine whose address is stored in the SOMError global variable. For more information about the SOMError replaceable routine, see “Replacing the SOMError Routine” on page 1-4.

The rest of this section will discuss the following topics on exceptions:

- Introduction to Exceptions
- The Environment
- Setting an Exception Value
- Getting an Exception Value
- Example Of Raising an Exception

This chapter documents General-use Programming Interface and Associated Guidance Information provided by SOMobjects.

Introduction to Exceptions

SOMobjects follows the CORBA model for exception handling. In this model the method caller receives error information back from the method invocation in a data structure called the Environment. This is different from the *catch/throw* model where an exception is implemented by a long jump or a signal.

CORBA defines two types of exceptions:

USER_EXCEPTION Explicitly declared in IDL files. Every method that returns a user exception contains a **raises** keyword listing the exceptions it may return.

SYSTEM_EXCEPTION Implicitly defined. Any method may return these exceptions without listing them on a **raises** keyword. System exceptions are sometimes called standard exceptions.

User Exceptions

In SOM Interface Definition Language, a method may be declared to return zero or more exceptions. Each type of exception has a name and, optionally, a struct-like data structure for holding error information. A method declares the types of exceptions it may return in a **raises** expression.

Below is an example IDL declaration of a BAD_FLAG exception, which may be *raised* by a checkFlag method, as part of a MyObject interface:

```
interface MyObject {
   exception BAD_FLAG {long ErrCode; char Reason[80]; };
   void checkFlag(in unsigned long flag) raises(BAD_FLAG);
};
```

An exception structure contains information to help the caller understand the nature of the error. The exception declaration can be treated like a struct definition: that is, whatever you can access in an IDL struct, you can access in an exception declaration. Alternatively, the structure can be empty, whereby the exception is just identified by its name.

The SOM Compiler will map the exception declaration in the above example to the following C language constructs:

```
typedef struct BAD_FLAG {
   long ErrCode;
   char Reason[80];
} BAD_FLAG;
#define ex_BAD_FLAG "MyObject::BAD_FLAG"
```

When an exception is detected, the checkFlag method must call the **SOMMalloc** function to allocate a BAD_FLAG structure, initialize it with the appropriate error information, and make a call to the **somSetException** function to record the exception value in the Environment structure passed in the method call. The caller, after invoking checkFlag, can check the Environment structure that was passed to the method to see if there was an exception and, if so, extract the exception value from the Environment.

System Exceptions

In addition to user-defined exceptions (those defined explicitly in an IDL file), there are several predefined exceptions for system run-time errors. A system exception can be returned on any method call. (That is, they are implicitly declared for every method whose class uses IDL call style, and they do not appear in any **raises** expressions.) Most of the predefined system exceptions pertain to Object Request Broker errors. Consequently, these types of exceptions are most likely to occur in DSOM applications.

Each of the standard exceptions has the same structure: an error code (to designate the subcategory of the exception) and a completion status code. For example, the NO_MEMORY standard exception has the following definition:

```
enum completion_status {YES, NO, MAYBE};
exception NO_MEMORY { unsigned long minor;
                      completion_status completed; };
```

The completion status value indicates whether the method was never initiated (NO), completed execution prior to the exception (YES), or the completion status is indeterminate (MAYBE).

Because all the standard exceptions have the same structure, file **SOMMVS.SGOSH.H(SOMCORBA)** included by **SOMMVS.SGOSH.H(SOM)** defines a generic **StExcep** typedef which can be used instead of the specific typedefs:

```
typedef struct StExcep {
     unsigned long minor;
     completion_status completed;
} StExcep;
```

The standard exceptions are defined in an IDL module called **StExcep**, in the file named **SOMMVS.SGOSIDL.IDL(STEXCEP)**, and the C definitions can be found in **SOMMVS.SGOSH.H(STEXCEP)**.

The Environment

The Environment is a data structure that contains environmental information that can be passed between a caller and a called object when a method is executed. For example, it is primarily used to return exception data to the client following a method call.

A pointer to an Environment variable is passed as an argument to method calls. The Environment typedef is defined in SOMMVS.SGOSH.H(SOMCORBA), and an instance of the structure is allocated by the caller in any reasonable way:

- on the stack (by declaring a local variable and initializing it using the **SOM_InitEnvironment** macro),
- dynamically (using the **SOM_CreateLocalEnvironment** macro), or
- by calling the **somGetGlobalEnvironment** function to allocate an **Environment** structure.

Setting an Exception Value

To set an exception value in the caller's **Environment** structure, a method implementation makes a call to the **somSetException** procedure:

```
void  somSetException (Environment *ev,
                       exception_type major,
                       string exception_name,
                       void *params);
```

where *ev* is a pointer to the **Environment** structure passed to the method, *major* is an **exception_type**, *exception_name* is the string name of the exception (usually the constant defined by the IDL compiler, for example, ex_BAD_FLAG), and *params* is a pointer to an (initialized) exception structure which must be allocated by **SOMMalloc**. The **exception_type** (for which *major* is defined) is:

```
typedef enum exception_type {
   NO_EXCEPTION, USER_EXCEPTION, SYSTEM_EXCEPTION,
exception_type_MAX=214783647
} exception_type;
```

The **somSetException** function expects the *params* argument to be a pointer to a structure that was allocated using **SOMMalloc** function. When **somSetException** is called, the client passes ownership of the exception structure to the SOM run-time environment. The SOM run-time environment will free the structure when the exception is reset (that is, upon next call to **somSetException**), or when the **somExceptionFree** function is called.

somSetException simply sets the exception value; it performs no exit processing. If there are multiple calls to **somSetException** before the method returns, the caller sees only the last exception value.

Getting an Exception Value

After a method returns, the calling client program can look at the **Environment** structure to see if there was an exception. The Environment struct is mostly opaque, except for an exception type field named **_major**:

```
typedef struct Environment {
   exception_type   _major;
...
} Environment;
```

If **ev._major != NO_EXCEPTION**, there was an exception returned by the call. The caller can retrieve the exception name and value (passed as parameters in the **somSetException** function call) from an **Environment** struct with the following functions:

```
string  somExceptionId (Environment *ev);
somToken  somExceptionValue (Environment *ev);
```

The **somExceptionId** function returns the exception name, if any, as a string. The **somExceptionValue** function returns a pointer to the value of the exception, if any, contained in the exception structure. If NULL is passed as the **Environment** pointer in either of the above calls, an implicit call is made to the **somGetGlobalEnvironment** function.

The **somExceptionFree** function frees any memory in the **Environment** associated with the last exception. This function does only a shallow **SOMFree** of the Environment's exception parameters. It does not walk the exception parameters, freeing any nested memory blocks.

You can also use the CORBA **exception_free** API to free the memory in an Environment structure.

File **SOMMVS.SGOSH.H(SOMCORBA)** included by (**SOMMVS.SGOSH.H(SOM)**) provides the following aliases for strict compliance with CORBA programming interfaces:

```
#ifdef CORBA_FUNCTION_NAMES
#define exception_id    somExceptionId
#define exception_value somExceptionValue
#define exception_free  somExceptionFree
#endif /* CORBA_FUNCTION_NAMES */
```

Example Of Raising an Exception

The following IDL interface for a MyObject object in a file called myobject.idl declares a BAD_FLAG exception, which can be raised by the *checkFlag* method:

```
interface MyObject {
   exception BAD_FLAG { long ErrCode; char Reason[80]; };
   void checkFlag(in unsigned long flag) raises(BAD_FLAG);
};
```

The SOM IDL compiler maps the exception to the following C language constructs, in **myobject.h**:

```
typedef struct BAD_FLAG {
   long ErrCode;
   char Reason[80];
} BAD_FLAG;
#define ex_BAD_FLAG "MyObject::BAD_FLAG"
```

A client program that invokes the *checkFlag* method might contain the following error handling code.

Note: The error checking code below lies in the user-written procedure, *ErrorCheck*, so the code need not be replicated through the program.

```
#include "som.h"
#include "myobject.h"
boolean ErrorCheck(Environment *ev);   /* prototype */
main()
{
   unsigned long flag;
   Environment ev;
   MyObject myobj;
   char     *exId;
   BAD_FLAG *badFlag;
   StExcep  *stExValue;
   myobj = MyObjectNew();
   flag  = 0x01L;
   SOM_InitEnvironment(&ev);
   /* invoke the checkFlag method, passing the Environment
      parameter */
   _checkFlag(myobj, &ev, flag);
   /* check for exception */
   if (ErrorCheck(&ev))
   {
      /* ... */
      somExceptionFree(&ev);   /* free the exception memory */
   }
   /* ... */
}
/* error checking procedure */
boolean ErrorCheck(Environment *ev)
{
   switch (ev._major)
   {
   case SYSTEM_EXCEPTION:
      /* get system exception id and value */
      exId      = somExceptionId(ev);
      stExValue = somExceptionValue(ev);
      /* ... */
      return(TRUE);
   case USER_EXCEPTION:
      /* get user-defined exception id and value */
      exId = somExceptionId(ev);
      if (strcmp(exId, ex_BAD_FLAG) == 0)
      {
         badFlag = (BAD_FLAG *) somExceptionValue(ev);
         /* ... */
      }
      /* ... */
      return(TRUE);
   case NO_EXCEPTION:
      return(FALSE);
   }
}
```

The implementation of the *checkFlag* method may contain the following error-handling code:

```
#include "som.h"
#include "myobject.h"
void checkFlag(MyObject somSelf, Environment *ev,
               unsigned long flag)
{
   BAD_FLAG *badFlag;
   /* ... */
   if ( /* flag is invalid */ )
   {
      badFlag = (BAD_FLAG *) SOMMalloc(sizeof(BAD_FLAG));
      badFlag->ErrCode = /* bad flag code */;
      strcpy(badFlag->Reason, "bad flag was passed");
      somSetException(ev, USER_EXCEPTION,
                      ex_BAD_FLAG, (void *)badFlag);
      return;
   }
   /* ... */
}
```

Chapter 3. Using SOM Trace Facilities

This section describes how to debug problems with SOM trace facilities.

This section consists of the following topics:

This chapter, except for "Writing Applications to Use SOM Trace Facilities" on page 3-4, documents Diagnosis, Modification or Tuning Information.

Capturing Trace Records

In order to capture SOM trace records into a trace data set, you must modify the trace level to indicate the level of tracing, start the particular SOM process, and then stop the process for the data to be saved. These steps are:

1. Modify the following environment variables. For descriptions of configuration file variables, see the *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

   - SOM_TraceLevel = n. This variable will establish the SOM trace level, with n being one of:

     0 Keeps SOM tracing off.

     1 Minimum trace data captured: error messages and exceptions.

     2 Capture above, plus communication messages.

     3 Capture above, plus entry/exit and calls to external functions.

     4 Capture above, plus status and miscellaneous messages.

     5 Capture above, plus error log records as captured by the error log facility (see Chapter 4, "Using the Error Log Facility" on page 4-1).

     This variable is set through an OS/390 OpenEdition environment variable: ENVAR(SOM_TraceLevel=n). However, in the case where this variable gets set for the SOM subsystem, SOM_TraceLevel is a configuration file variable and is set in the somras stanza of the SOM configuration file (SOMENV).

   - MVSTraceLog = <high level qualifier> of the trace data set.

     This value must be alphanumeric (starting with an alphabetic character), from 1 to 8 characters in length. This variable must be specified to get trace records written to a data set. If this variable is not specified, tracing will be done, but only to an in-memory trace buffer.

   - MVSTraceLogSize = *nnnn ttt*, where *nnnn* is an integer and *ttt* is either CYL or TRK. (Note: There must be a space to separate *nnnn* from *ttt*.) This variable will establish the size of trace data set.

2. Start the process (SOM subsystem or server or other SOM application). Note that the trace data set is established during the initialization of the process. The data set name is displayed in SOM message GOS20007I.
3. Stop the process. This will close the trace data set and make it available for displaying the trace data.

Note: The trace data is captured during processing of the address space. If the trace data set fills, the tracing continues at the top of data set, with the trace data wrapping. The oldest trace records are overlaid with more recent trace data.

Activating Communication Traces

To activate communication traces, do the following:

- Set SOM_tracelevel to 2 or greater (for example, `SOM_tracelevel=3`)
- (Optional) Within the [somd] stanza of the configuration file, set the value of SOMDCOMMDEBUGFLAG=*n*, where *n* is as follows:

  1 enables a trace of TRANSPORT information. TRANSPORT information is data associated with the communication socket, such as Create Listener and Sending Message.

  2 enables a trace of DATASTREAM information. DATASTREAM information is the actual data sent through a distributed request.

  4 enables a trace of DISPATCH information. DISPATCH information is the dispatch of a distributed method request by a DSOM server.

  Note that you can combine the values. For example, if you want to display DATASTREAM and TRANSPORT information, set the following value: `SOMDCOMMDEBUGFLAG=3`. The default is 7 when SOM_tracelevel is 2 or greater. Otherwise, the default is 0.

  See *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* for a description of the configuration file.

Displaying Captured Trace Records

You can use the MVS Interactive Problem Control System (IPCS) CTRACE command to view SOM trace data.

To display the SOM trace data, determine the name associated with the trace data. The system establishes the SOM trace under the following names:

```
component name = SYSDSOM
sublevel name  = <jobname><ASID>   (i.e., jobname concatenated with ASID)
```

If the SOM server is started with a jobname of SOMDSVR and its ASID is 01C5 (hexadecimal), the sublevel name is SOMDSVR01C5. If you don't know the sublevel name, you can use the IPCS CTRACE QUERY subcommand to display both the component name and the sublevel name. The syntax of the IPCS CTRACE QUERY subcommand is:

```
CTRACE QUERY DSN(data-set-name-containing-trace)
```

Display the SOM trace data with the following IPCS CTRACE subcommand:

```
CTRACE COMP(SYSDSOM) SUB((sublevel-name))       +
     <Report Type Keywords>                     +
     DSN(data-set-name-containing-trace)        +
     <filter options>                           +
     [OPTIONS((component options))]
```

Some of the report type keywords that you may specify are SHORT, SUMMARY, and FULL.

The optional component options for SOM are:

SKIPID	Specify to omit display of the jobname, ASID and thread ID.
SHORTFORM	Specify to get a short form of the display, where each trace function is displayed on a single line. SHORTFORM is the default (versus LONGFORM).
LONGFORM	Specify to get a long form of the display, where each trace function has (potentially) multiple trace elements, each on a separate line.

IBM recommends using the following parameters to display the trace data:

```
CTRACE COMP(SYSDSOM) SUB((sublevel-name)) FULL   +
     DSN(data-set-name-containing-trace)        +
     OPTIONS((SKIPID,SHORTFORM))
```

Combining the Output of Multiple Traces

SOM trace records from multiple traces may be combined into a single display. Using IPCS MERGE subcommand processing, multiple traces may be displayed in timestamp order. In this manner, you can analyze processing between a server and a client, for example.

The syntax for using MERGE is similar to that of CTRACE. You specify MERGE, followed by all of the CTRACE invocations, and ended with MERGEEND. For example:

```
MERGE
CTRACE COMP(SYSDSOM) SUB((sublevel-name-1)) FULL +
     DSN(data-set-name-1) OPTIONS((SKIPID))
CTRACE COMP(SYSDSOM) SUB((sublevel-name-2)) FULL +
     DSN(data-set-name-2) OPTIONS((SKIPID))
[CTRACE ...]
...
MERGEEND
```

Displaying SOM Trace Data from an OS/390 Dump

The SOM trace data may be displayed from an OS/390 dump data set using the same tools as for processing from a trace data set. The syntax for the CTRACE subcommand is the same as above; the difference is that you specify the MVS dump data set name rather than the trace data set name.

Note that if you expect to gather trace data only via the OS/390 dump, you may omit the MVSTraceLog environment variable when starting the particular process. As long as the SOM_TraceLevel environment variable is a non-zero value, SOM trace data will be captured into in-memory trace buffers. These trace buffers will get captured along with other relevant information when an OS/390 dump is taken for the particular address space.

You may combine the displays of the trace data set with the displays of the OS/390 dump using the set of MERGE and CTRACE subcommands. Specify the appropriate data set for each of the CTRACE invocations.

Sample JCL to Display SOM Trace Data

Figure 3-1 shows sample JCL that displays SOM trace data. Note that the JCL uses an IPCS dump directory (in VSAM data set "userid.DUMP.DIR") that must be allocated before you run the JCL. See *OS/390 MVS IPCS Commands* for information about initializing a dump directory.

Note: The user must specify:

1. job card information
2. userid on IPCSDDIR DD
3. sublevel-name
4. trace data set name

```
//SHOWTRC  JOB <job card info>
//JOBLIB DD DSN=SOMMVS.SGOSLOAD,DISP=SHR
//       DD DSN=SYS1.MIGLIB,DISP=SHR
//IPCS  EXEC  PGM=IKJEFT01,REGION=1000M
//IPCSDDIR DD DSN=userid.DUMP.DIR,DISP=(OLD,KEEP)
//IPCSPARM DD DSN=SYS1.PARMLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//IPCSTOC  DD SYSOUT=*
//IPCSPRNT DD SYSOUT=*
//* ------------------------------------------------------------
//SYSTSIN DD *
IPCS NOPARM
CTRACE COMP(SYSDSOM) FULL SUB((sublevel-name))              +
  DSN('trace.data.set.name') OPTIONS((SKIPID,SHORTFORM))
/*
```

Figure 3-1. Example of Trace JCL. This example shows how you would display SOM trace data.

Writing Applications to Use SOM Trace Facilities

This section documents General-use Programming Interface and Associated Guidance Information provided by SOMobjects.

Applications may be written to use this SOM tracing facility. A call to the MVSTrace method is done to record the trace event.

SOM trace data is captured into the following trace events:

```
 Event                                      SOM_TraceLevel
 ----------------------------------------   --------------
 1 - New trace buffer                       1
 2 - Error message                          1
 3 - Communication message                  2
 4 - Entry to method                        3
 5 - Exiting method                         3
 6 - About to call external function        3
 7 - Return from call to external function  3
 8 - Status message                         3
 9 - Miscellaneous                          4
10 - SOM error log records                  5
```

The above trace data table lists all events captured in SOM tracing. Events 1 through 10 are used by SOMobjects. Events 4, 5 and 9 may be used by an application program to help debug user programs. Note that different levels of trace keys are associated with different levels of trace processing. As indicated in the trace table, a SOM_TraceLevel of 1 will result in tracing only error messages (key 1), a SOM_TraceLevel of 2 will result in tracing communication messages and error messages, and a SOM_TraceLevel of 5 will result in tracing everything.

The interface for the MVSTrace method call and the variables for the trace events are contained within the publicly available header data set. The header which contains these definitions is SOMMVS.SGOSH.H(MVSTRACE). Include this member in your C program to get the appropriate trace event variables and the prototype for the MVSTrace method to invoke.

Chapter 4. Using the Error Log Facility

SOMobjects supports an error log to record exceptions and error conditions that may occur within the SOMobjects services.

Most of the data recorded in the error log is formatted to help you debug new applications. The remaining data is not formatted and can be used by IBM service to help diagnose more complicated problems.

The error log records only exceptions and errors that occur within code shipped with SOMobjects. If you wish to record error information from the frameworks and applications that you develop, you must provide your own facility to do this.

The error log is implemented as a wrapping log file. Once it has filled up, the oldest entries are destroyed as new entries are added. All processes on the same system should be set up to share a single error log file for greater ease in solving any multi-process interaction problems that might occur. If you are using SOM at multiple locations, you have multiple error logs.

Error log files are located in the file identified by the SOMErrorLogFile configuration variable, under the [somras] stanza.

The rest of this section will discuss the following topics:

- "Configuring the Error Log"
- "Using The Error Log" on page 4-2
- "Understanding Error Log Entries" on page 4-3
- "Locating the Correct Log File" on page 4-5

This chapter documents Diagnosis, Modification or Tuning Information.

Configuring the Error Log

There are four variables in the configuration file, under the [somras] stanza, which you can set to customize the operation of the error log.

Name of the Error Log File

The **SOMErrorLogFile** variable controls the name of the error log file. The default setting of SOMErrorLogFile is SOMERROR.LOG. Prefix the specification with // to indicate a data set. For a fully qualified data set name, add quotes around the name (after the //, if any); for a non-quoted data set specification, the userid associated with the process is prepended to the specification.

Note: Your application must have RACF Write access to the error log.

Size of the Error Log

The **SOMErrorLogSize** variable controls the maximum size of the error log file. The default size is 128 which lets the error log file grow to 128 kilobytes before it begins to wrap. The default size allows room for several hundred average log entries.

Type of Information To Record

The **SOMErrorLogControl** variable lets you filter the information to record in the Error Log. This includes severity or type of error logged. Examples include:

WARNING This is a default setting. If specified, records classified as “warning” are written to the error log.

ERROR This is a default setting. If specified, records classified as “error” are written to the error log.

MAPPED_EXCEPTION This is a default setting and is used to select whether to record a message each time an object service maps an exception into a different exception. This often occurs as an exception raised by an object service ripples back up through the object system to the application.

INFO This is not a default setting and must be individually selected.

The default setting of SOMErrorLogControl is

```
WARNING ERROR MAPPED_EXCEPTION
```

To specify multiple values, as the default setting does, separate each value by one or more blank spaces.

Display Error Messages

The **SOMErrorLogDisplayMsgs** variable controls whether to also display the error message to the standard output device each time an Error Log entry is made. The displayed messages do not include any extended log data collected for service personnel. The default setting of SOMErrorLogDisplayMsgs is YES. The default setting logs errors and displays the optional messages. This setting is helpful while you are debugging new applications.

Using The Error Log

Much of the data in the Error Log is formatted. You can use a text editor to display the contents of the error log file. The top lines of the file contain the name of the host system that this error log file is from, the operating system, and the number of the last log entry written in the file.

To find the last error log entry in the file look for the LASTENTRY: *nnnn* line in the information at the top of the file. You can then use the search feature of your text editor to locate this entry *nnnn* in the log file. Because the error log wraps around, this entry may be anywhere in the file.

Using a text editor for an HFS file, the user sees multiple lines. But using ISPF edit (or browse), the user sees a single record. These records are logically divided, with a character of ‘15’ hexadecimal marking the divisions.

Understanding Error Log Entries

Each error log entry starts with a message area and may contain extended log data. The message area contains the following free-form fields:

- The error log entry number enclosed in brackets. Error log entry numbers can range from 0 to 9999. When the error log entry number reaches 9999 it wraps back to 0 and starts over again.
- A date and time stamp telling the system date and time when the error log entry was made.
- PID: 0X*nnnn* is the process identifier of the program that made the error log entry call.
- TID: 0X*nnnn* identifies the thread, within the process, that made the call.
- ASID: 0Xaaaa identifies the address space identifier (ASID) of the process that made the error log entry call.
- JobName: identifies the jobname of the process that made the error log entry call.
- Request from *clientX* on *hostY* identifies the client name and host name where the operation request originated. Normally this information is retrieved from the Principal object that accompanies each DSOM request. If the Principal object is not available, this portion of the message may say Request from unknown client on unknown host. You can use this information to help debug servers where requests are coming from different hosts and clients.

The second part of an error log entry is the optional extended log data. If included, this part of the entry contains hexadecimal data followed by an EBCDIC translation of the data. Most of this data is useful only to service personnel. However, some of the action lists in Chapter 7, "SOMobjects Error Codes" on page 7-1 may point out information in this data that will be useful to you.

A complete error log entry that contains both a message and extended log data may look like the following.

```
{0464} Wed Sep  4 15:06:20 1996  PID:    0X90067 TID:          0  ASID: 0X0052 JobName: KEVANS7
GOS20001E Raised SYSTEM_EXCEPTION INITIALIZE with severity ERROR at MVSDSOM.S3DR5.SOMD.C(SOMDFENC):1800.
Request from clientX on hostY.
Error code is 30088[SOMDERROR_NamingNotActive].
```

If the extended log data is quite large, you may see continuation blocks that begin with {+*nnnn*}, where *nnnn* is the same number used at the beginning of the log entry. The plus symbol (+) indicates this is a continuation block for the same long entry. A continuation block for the above example would begin:

```
{+0464} Wed Sep  4 15:06:20 1996  PID:    0X90067 TID:          0  ASID: 0X0052 JobName: KEVANS7
```

It is possible for one log entry to have several continuation blocks.

The Standard Error Messages

The following list shows the four basic message types that may be logged. The examples do not include the optional extended log data.

“Raised Exception” Message Type

```
{1234} Mon Oct 21 16:58:53 1996 PID: 0X1000003 TID:     0 ASID: 0C0029  JobName: DSOM
GOS20001E raised SYSTEM_EXCEPTION UNKNOWN with severity ERROR at MVSDSOM.S3DR5.SOMDCOMM.C(CALLSIOP):4282.
Request from clientX on hostY.
Error code is 30003[SOMDERROR_InvalidProtocolInformation].
```

This type of message is logged by a SOM service when it raises a new exception (that is, records an exceptional condition into the environment structure). This message includes an error code. See Chapter 7, “SOMobjects Error Codes” on page 7-1 to determine its meaning and actions to take to correct the problem. This type of message also contains a level indicator. See “Level of Errors” on page 4-5 for more information.

“Mapped Exception” Message Type

```
{1240} Thu Oct 24 20:54:35 1996 PID 0X4000003 TID:       0 ASID: 0X0016  JobName: SOM@CFG
GOS20002E Mapped USER_EXCEPTION ::SysAdminException::ExExists to SYSTEM_EXCEPTION
::StExcep::PERSIST_STORE at MVSDSOM.S3DR5.CORE.C(SOMAPUTL):559.
Request from clientX on hostY.
New Error code is 00001[Exception_Mapped].
```

This type of message indicates that an SOM service received an exception from a sub-service method call and has mapped the original exception into a new exception. This mapping is often required because services can raise only User Exceptions listed on their interface descriptions or Standard CORBA Exceptions. You may follow the trail of mapped exceptions to understand if the exception received by the application code has been re-mapped from the object service exception that was originally raised. If re-mapping has occurred, the action list for the original exception, that is, the one with message type (1) above, may often contain more helpful information for resolving the problem.

“Abnormal Termination” Message Type

```
{1250} Thur Oct 24 21:45:42 1996 PID:  0X6000003 TID:      0 ASID: 0X002A  JobName: DSOMN
GOS20003E Process abnormally terminated at MVSDSOM.S3DR5.OSSVR.C(SOMOSSVR):201.
Request from clientX on hostY
Error code is 57009[SOMOS_Impl_Is_Ready_Failed].
```

This type of message indicates an error situation that prevented the SOM service from operating reliably.

“Service Data” Message Type

```
{1345} Fri Oct 25 10:15:21 16691 PID:  0X4000003 TID:      0 ASID: 0X0041  JobName: SOM@CFG
GOS20004E Service data collected with severity ERROR at MVSDSOM.S3DR5.SOMBT.C(BDBA):638.
Request from clientX on hostY.
Error code is 0[UNDEFINED].
```

The action lists in Chapter 7, “SOMobjects Error Codes” on page 7-1 might provide useful information. This type of message might indicate that the object service logged data for use by IBM Service.

Extended Error Messages

Each SOM service may also supply some unique extended error messages. Extended error messages always begin with the text from one of the four standard message types and then append service specific information to the end of the message.

The extended message below illustrates an extension of "service data" message type.

```
{1345} Fri Oct 25 10:15:21 1996  PID:0X4000003 TID:        0 ASID: 0X0041  JobNAme:SOM@CFG
GOS20004E Service data collected with severity ERROR at MVSDSOM.S3DR5.SOMBT.C(BDBA):638.
Request from clientX on hostY.
Error code is 0[UNDEFINED].
GOSB0005E IDCAMS failed to create file "SOMMVS.impl.db".
```

In this extended error message example *'GOSB0005E IDCAMS failed to create file “SOMMVS.impl.db”.'* is the SOM service unique extension that was added to a standard message.

Level of Errors

Some error log entries contain an indication of the error level. The following error levels are used.

- INFO indicates a problem which is usually handled within the SOMobjects code and probably does not affect your application.
- WARNING indicates a problem that may cause your application to not function properly. Warning messages cannot be ignored. You must resolve these errors for your application to function properly.
- ERROR indicates a problem that most likely prevents your application from functioning properly. In some cases, SOM may recognize particular errors and continue to process normally.

Locating the Correct Log File

In order to use the Error Log effectively you must first determine which systems' log file you need to look at. If your application invokes methods on server processes running in remote systems, you may have to examine the log files on the server systems to determine what error occurred. It may be helpful to keep SOMErrorLogDisplayMsgs set to YES on each of the server systems to display error messages while you are debugging your new application. This lets you determine which system originally raised an exception. Once you have identified the system which raised the exception, you can look up the error code in Chapter 7, “SOMobjects Error Codes” on page 7-1 and use the action list to help you resolve the problem.

Chapter 5. SOMobjects Messages

This chapter documents Diagnosis, Modification or Tuning Information.

This chapter provides information about messages issued by SOMobjects.

There are two types of SOMobjects messages:

- SOM Subsystem messages for the system programmer

  The system messages begin in " SOMobjects Subsystem Messages." They occur on the operator system console.

- Application programmer messages

  The rest of the messages in this book are application programmer messages of SOMobjects components and begin in "BTree Messages" on page 5-7.

The format of the system messages is slightly different than those for the application programmer. The following is the format for the SOM subsystem messages:

```
GOSnnns
```

where GOS is the SOM product identifier, *nnn* is a serial number that identifies each message, and *s* is a type code that is one of the following:

E	Error
I	Information
S	Severe
W	Warning

SOMobjects Subsystem Messages

GOS006I SOM/MVS SUBSYSTEM JOBNAME *jobname* IS NOT VALID.

Explanation: The SOMobjects procedure name, which is used as the name of the subsystem, is not a valid 1-4 character name.

In the message text:

jobname
The name of the SOMobjects job or started procedure.

System Programmer Response: Ensure that the SOMobjects job name/procedure name is a valid subsystem name (1-4 characters).

System Action: The SOMobjects subsystem ends.

Detecting Module:
SOM

GOS007I SOM/MVS *ssname* INPUT COMMAND PREFIX IS NOT VALID. DEFAULT PREFIX USED.

Explanation: A command prefix value that was specified as input to the SOMobjects subsystem *ssname* is not valid. The prefix value is either greater than 8 characters, begins with an invalid symbol, or contains an imbedded blank.

In the message text:

ssname
is the name of the SOMobjects subsystem.

Operator Response: If the default command prefix is unacceptable, enter a CANCEL command prefixed by the *ssname* to cancel the SOMobjects subsystem immediately. Then if the bad command prefix was entered by you as a parameter on the START command when you first started SOMobjects, re-enter the START command to restart the SOMobjects subsystem with a valid command prefix value.

If the bad command prefix was not specified by you on the START command, notify your system programmer.

System Programmer Response: If the bad command prefix was specified as a parameter in the job procedure that was used to start the SOMobjects subsystem, correct the command prefix parameter in the procedure so the next time the subsystem is started, the desired command prefix will be used.

System Action: SOMobjects subsystem initialization continues. The specified command prefix value is ignored and as a default, the subsystem name *ssname* is used as the command prefix.

Detecting Module:
SOM

GOS008I SOM/MVS *ssname* **COMMAND PREFIX IS** *cmdprefix*

Explanation: The command prefix that is being used by the SOMobjects subsystem *ssname* is *cmdprefix*. All commands entered for the subsystem should begin with this prefix.

In the message text:

ssname
The name of the SOMobjects subsystem.

cmdprefix
is a 1-8 character command prefix.

Operator Response: Whenever you enter one of the commands supported by the SOMobjects subsystem, you must begin the command with the indicated command prefix (e.g., *cmdprefix* STATUS).

System Action: SOMobjects subsystem initialization continues.

Detecting Module:
SOM

GOS009I SOM/MVS *ssname* **INITIALIZATION COMPLETE.**

Explanation: The SOMobjects subsystem *ssname* has completed its initialization.

In the message text:

ssname
is the name of the SOMobjects subsystem.

System Action: The SOMobjects subsystem continues processing. It is now ready to accept distributed SOM requests.

Detecting Module:
SOM

GOS010I SOM/MVS *ssname* **ENDED.**

Explanation: The SOMobjects subsystem *ssname* has completed shutting down.

In the message text:

ssname
is the name of the SOMobjects subsystem.

System Action: The SOMobjects subsystem space ends.

Detecting Module:
SOM

GOS011I SOM/MVS *ssname* INITIALIZATION FAILED.

Explanation: The SOMobjects subsystem *ssname* could not be successfully initialized. The specific initialization error is indicated by a SOMobjects error message or abend that was issued just prior to this message.

In the message text:

ssname
is the name of the SOMobjects subsystem.

System Programmer Response: Examine the SYSLOG to determine the error message or abend that caused initialization to fail and then correct the condition that caused the error.

System Action: The SOMobjects subsystem ends.

Detecting Module:
SOM

GOS012I SOM/MVS *ssname* ALREADY ACTIVE.

Explanation: One instance of the SOMobjects subsystem *ssname* already exists. Starting more than one SOMobjects subsystem is not allowed.

In the message text:

ssname
The name of the SOMobjects subsystem.

Operator Response: If you were restarting the SOMobjects subsystem after an error situation, ensure that the first instance of the subsystem ends before issuing the START command.

System Action: The SOMobjects subsystem that issued this message ends.

Detecting Module:
SOM

GOS013I SOM/MVS *ssname* IS NOT A VALID SUBSYSTEM.

Explanation: The SOMobjects subsystem *ssname* is not a defined subsystem name known by the system.

In the message text:

ssname
The name of the SOMobjects subsystem.

System Programmer Response: Ensure the SOMobjects subsystem name *ssname* is defined as a valid subsystem name in the IEFSSNxx SYS1.PARMLIB member that was used to IPL the system.

System Action: The SOMobjects subsystem ends.

Detecting Module:
SOM

GOS019I INITIALIZATION COMPLETE FOR SOM/MVS SERVER *server_name* WITH ALIAS *alias_name*

Explanation: A SOMobjects server has completed initialization. The ALIAS for this server is *aliasname*.

In the message text:

server_name
The name of the SOMobjects server, which is specified on DSOM server activation commands. See *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* for descriptions of those commands.

alias_name
The alias of the SOMobjects server.

System Action: The SOMobjects server continues processing. It is ready to accept requests from distributed SOM clients.

Detecting Module:
SOM

GOS021I SOM/MVS *ssname* SUBSYSTEM FUNCTIONS DISABLED.

Explanation: In the process of ending either normally or abnormally, the SOMobjects subsystem *ssname* disabled the subsystem functions it normally provides for jobs that use SOM services via the SOMobjects subsystem.

In the message text:

ssname
is the name of the SOMobjects subsystem.

System Action: The SOMobjects subsystem ends. SOM requests will no longer be processed. Any usage of servers, clients, and SOM utilities will be affected.

Detecting Module:
SOM

GOS022I SOM/MVS *ssname* SUBSYSTEM FUNCTION DISABLEMENT FAILED.

Explanation: In the process of ending either normally or abnormally, the SOMobjects subsystem *ssname* attempted to disable the subsystem functions it normally provides for jobs that use SOM services via the SOMobjects subsystem. However, a failure occurred that prevented the subsystem from completely disabling all the subsystem functions it supports.

In the message text:

ssname
The name of the SOMobjects subsystem.

System Action: The SOMobjects subsystem ends. Any jobs, managed by the subsystem shown in the message text, that try to use SOM services might abnormally end.

Detecting Module:
SOM

GOS023I SOM/MVS SUBSYSTEM *ssname* ALTERED TO USE THE PRIMARY SUBSYSTEM.

Explanation: The SOMobjects subsystem is to be started only under the primary subsystem. The system issues this message if your installation specifies the SOMobjects subsystem initialization routine, GOSAMSSI, on the initialization statement for the SOMobjects subsystem in the IEFSSNxx parmlib member. The SOMobjects subsystem initialization routine will always force the specified SOMobjects subsystem is to be initialized under only the primary subsystem. This message is written to hardcopy only.

In the message text:

ssname
The name of the SOMobjects subsystem.

System Action: System initialization continues.

Detecting Module:
SOM

GOS024I SOM/MVS *ssname cmdname* COMMAND ERROR -- SPECIFIED KEYWORD IS NOT VALID.

Explanation: The command entered, *cmdname*, specified a keyword that is not supported.

In the message text:

ssname
The name of the SOMobjects subsystem.

cmdname
The name of the command that was entered.

System Action: The SOMobjects subsystem stops processing the command.

Operator Response: If the keyword was misspelled, enter the command again with the correct keyword specified. Otherwise, if the keyword should not have been specified, enter the command again without the unsupported keyword.

Detecting Module:
SOM

GOS025I MVS/DSOM *ssname cmdname* **COMMAND ERROR -- NOT AUTHORIZED TO ISSUE COMMAND.**

Explanation: The operator/console is not authorized to enter the command, *cmdname*.

In the message text:

ssname
The name of the SOMobjects subsystem.

cmdname
The name of the command that was entered.

Operator Response: Contact your installation's security administrator to ensure both you and the console are properly authorized to enter the command that you were attempting.

System Action: The SOMobjects subsystem stops processing the command.

Detecting Module:
SOM

GOS026I MVS/DSOM *ssname* **STOP INITIATED.**

Explanation: In response to a STOP command, the SOMobjects subsystem *ssname* has initiated stop processing.

In the message text:

ssname
The name of the SOMobjects subsystem.

System Action: The SOMobjects subsystem quiesces its processing and ends processing for servers that are in running. When all server activity ends, the SOMobjects subsystem ends normally.

Detecting Module:
SOM

GOS027I *hh.mm.ss ssname* **HELP INFO**

```
MVS/DSOM COMMAND SYNTAX:
 ... STOP DAEMON
 ... CANCEL DAEMON
 ..... Not Supported
```

Explanation: This message is a multiline message issued in response to the HELP command. It displays a summary of the syntax for the commands supported by the SOMobjects subsystem shown in the message text.

In the message text:

hh.mm.ss
The time in hours (00-23), minutes (00-59), and seconds (00-59).

ssname
The name of the SOMobjects subsystem.

System Action: Processing continues.

Detecting Module:
SOM

GOS031I SOM/MVS *ssname* INITIALIZATION OF DAEMON SUBTASK FAILED.

Explanation: The SOMobjects subsystem *ssname* could not properly initialize the SOMobjects subsystem. More error information may be available from the SYSPRINT output, if available, or from the TRACE log, if available.

In the message text:

ssname
is the name of the SOMobjects subsystem.

Operator Response: Contact the system programmer.

System Programmer Response: Examine the SYSLOG or the job's log or the SYSPRINT data set for the job to determine the error message or abend that caused initialization to fail. Analyze the TRACE log, if one was requested. Ensure that the correct data sets are specified for the GOSRTL1 DD. Correct the condition that caused the error.

System Action: The SOMobjects subsystem ends.

Detecting Module:
SOM

GOS032I SYSDSOM COMPONENT TRACE FAILED.
DIAG1: *nnnnnnnn xxxxxxxx*

Explanation: The SYSDSOM (SOMobjects) component tried to initialize component tracing using default options. The system is now running without component tracing for SYSDSOM.

In the message text:

nnnnnnnn
Used by IBM for problem determination.

xxxxxxxx
Used by IBM for problem determination.

Operator Response: See the operator response for the component trace messages (prefix ITT) accompanying this message.

System Programmer Response: See the system programmer response for component trace messages (prefix ITT) accompanying this message.

Detecting Module:
SOM

System Action: Initialization continues without component tracing for SYSDSOM. The system issues component trace messages (prefix ITT) explaining the problem.

GOS033I DSOM TRACE REQUEST FAILED. OPTIONS ARE NOT ALLOWED.

Explanation: The system rejected the request to trace the system object model (SOM). The TRACE command specified options, but options are not allowed.

Operator Response: Enter the TRACE command again without specifying any options.

System Action: The system rejects the request to trace SOM.

Detecting Module:
SOM

GOS041I SOM/MVS *ssname* WAITING FOR SERVERS TO END.

Explanation: In the process of ending either normally or abnormally, the SOMobjects subsystem *ssname* must wait for distributed SOM (DSOM) servers to end before the subsystem is ended.

In the message text:

ssname
The name of the SOMobjects subsystem.

System Action: The SOMobjects subsystem will wait for the servers to end. If the servers do not end, the SOMobjects subsystem stops waiting and finishes ending. After that time, servers may end abnormally.

Detecting Module:
SOM

GOS042I SOM/MVS *ssname* WAITING FOR OMVS TO START.

Explanation: In the process of initialization, the SOMobjects subsystem *ssname* must wait for the OpenMVS address space to initialize before completing subsystem initialization.

In the message text:

ssname
The name of the SOMobjects subsystem.

System Action: The SOMobjects subsystem waits for the OMVS address space to initialize.

Detecting Module:
SOM

Message Format for SOMobjects Components

The messages for the SOMobjects components are in the following format:

```
GOS xyyyyz
 |  | |   |
 |  | |   └── Type of message
 |  | |         I - Informational
 |  | |         W - Warning
 |  | |         E - Error
 |  | |
 |  | └── Four-digit message identifier
 |  |
 |  └── Component identifier
 |        B - BTree messages
 |        C - Compiler messages
 |        D - Distributed SOMobjects (DSOM) messages
 |        J - Core Services messages
 |        K - Kernel messages
 |        N - Naming Service messages
 |        O - Object Services (OS) server messages
 |        V - Event Management Framework messages
 |        Y - Security Services messages
 |        Z - Configuration messages
 |        2 - SOMRAS Framework messages
 |        4 - SOM Collection Classes messages
 |
 └── The product identifier
```

BTree Messages

Following are the texts, explanations, and user responses for messages issued by the BTree framework. These messages appear in the program listing.

GOSB0001E I/O error occurred during *string1* processing, file \"*string2*\".

Explanation: An I/O error occurred while attempting to process the indicated file. If the file identifier ends in either ".SOMSYSIN" or ".SOMSYSPR", then the file is a sequential dataset. Otherwise, the indicated file is a VSAM file.

User Response: Insure that the high-level-qualifier (that is, file path) is READ/WRITE accessible by the program, and that there is data and directory space available for files. The high-level-qualifier originates from either the SOMDDIR variable (common case) or from the Persistence framework's somPersistenceBTREE_PID_BTREE class (method set_datastore_name). The system returned an "errno" value which is also displayed with results from the strerror() function. Reference the indicated system message for further information. If you need to contact IBM, please provide complete details from this message and from message GOSB0008I which follows.

GOSB0002E Could not open file \"*string1*\".

Explanation: The indicated file could not be opened for I/O. If the file identifier ends in either ".SOMSYSIN" or ".SOMSYSPR" then the file is a sequential dataset. Otherwise, the indicated file is a VSAM file.

User Response: Insure that the high-level-qualifier (that is, file path) is READ/WRITE accessible by the program, and that there is data and directory space available for files. The high-level-qualifier originates from either the SOMDDIR variable (common case) or from the Persistence framework's somPersistenceBTREE_PID_BTREE class (method set_datastore_name). The system returned an "errno" value which is also displayed with results from the strerror() function. Reference the indicated system message for further information. If you need to contact IBM, please provide complete details from this message and from message GOSB0008I which follows.

GOSB0003E Could not allocate file \"*string1*\" to \"DD:*string2*\".

Explanation: The indicated file was created successfully, but could not be allocated. It is likely the DD name is already allocated. Note that the IDCAMS utility is used to create VSAM files. IDCAMS requires the use of an input dataset allocation (DD:SYSIN by default) and an output dataset allocation (DD:SYSPRINT by default). SOM/MVS overrides the defaults with DD:SOMSYSIN for input, DD:SOMSYSPR for output to/from IDCAMS. These ddnames must be free and available for use.

User Response: Free the indicated DD name and re-try the operation.

GOSB0004E Could not load IDCAMS utility.

Explanation: The IDCAMS utility is required to create VSAM files. The call to fetch("IDCAMS") returned a null function address.

User Response: Insure IDCAMS is available to the system. Contact IBM if necessary.

GOSB0005E IDCAMS failed to create file \"*string1*\".

Explanation: The IDCAMS encountered an error while attempting to create the indicated VSAM file.

User Response: See the file <file_id>.SOMSYSPR for the IDCAMS results. Contact IBM if necessary.

GOSB0008I errno = *string1*, __amrc.__code.__error = *string2*, __amrc.__RBA = *string3*, __amrc.__last_op = *string4*

Explanation: This message appears immediately following messages GOSB001E or GOSB0002E. This message contains data which may aid in I/O error diagnosis, including the common "errno" value and relevant fields of the *_amrc* structure which commonly contains results of I/O operations.

User Response: Insure that the high-level-qualifier (that is, file path) is READ/WRITE accessible by the program, and that there is data and directory space available for files. The high-level-qualifier originates from either the SOMDDIR variable (common case) or from the Persistence framework's somPersistenceBTREE_PID_BTREE class (method set_datastore_name). The system returned an "errno" value which is also displayed with results from the strerror() function. Reference the indicated system message for further information. If you need to contact IBM, please provide complete details from this message.

Compiler Messages

Following are the texts, explanations, and user responses for the messages issued by the SOMobjects compiler.

SOMobjects returns one of the messages described in this section when it finds an error in the most recently completed SOMobjects compile. These messages appear in the program listing.

GOSC0001E Function name too long.

Explanation: A function name that you specified in the IDL exceeded the maximum length of 255 characters.

User Response: Shorten the function name and recompile the IDL.

GOSC0002E Comment exceeded comment buffer. Use -C option.

Explanation: A comment that you specified in the IDL exceeded the maximum length of 49,152 characters.

User Response: Shorten the comment or divide it into two comments and recompile the IDL.

GOSC0003E Function prototype too long.

Explanation: A function prototype name that you specified in the IDL exceeded the maximum length of 2048 characters.

User Response: Correct the function prototype name and recompile the IDL.

GOSC0005E No function name found in the following prototype: *string*

Explanation: A function prototype that you specified in the IDL did not contain a function name.

User Response: Specify a function name in the function prototype and recompile the IDL.

GOSC0006E Unexpected end of file encountered.

Explanation: Most likely, the compiler encountered an unexpected end delimiter in the IDL. There might be a mismatching bracket, conditional statement, or end of comment

User Response: Correct the mismatching delimiter and recompile the IDL.

GOSC0007E Syntax error, probably a missing parenthesis in a method prototype.

Explanation: IDL contained a syntax error. The cause of the syntax error might be a missing parenthesis.

User Response: Correct problem and resubmit job.

GOSC0008E Syntax error, probably a missing angle bracket in a method prototype.

Explanation: IDL contained a syntax error. The cause of the syntax error might be a missing angle bracket.

User Response: Check the syntax of the IDL. Ensure that all brackets match in the syntax. Then recompile the IDL.

GOSC0009E Syntax error. Probably a missing close quote in a method prototype.

Explanation: IDL contained a syntax error. The cause of the syntax error might be a missing right quotation mark.

User Response: Check the syntax of the IDL. Ensure that all quotation marks match in the syntax and recompile the IDL.

GOSC0010W File not found; creating new *string* file.

Explanation: The compiler could not find a file.

User Response: Check that the new file is acceptable.

GOSC0011W File is readonly, cannot be revised.

Explanation: A binding data set is readonly.

User Response: Check the RACF authorization of the data set.

GOSC0012I File has been updated.

Explanation: The compiler updated a binding file

User Response: None required.

GOSC0013I Changes stored in file *string*.

Explanation: The compiler stored changes in a binding file.

User Response: None required.

GOSC0014E Error: Insufficient memory.

Explanation: The compiler could not obtain the memory needed to create the required binding file.

User Response: In the JCL for the job, specify a larger region size on the REGION parameter. If that action does not work, contact the IBM Support Center.

GOSC0020E Cannot open Symbols file \"*string*\".

Explanation: The compiler could not open the emitter framework file which is input to the NEWEMIT utility

User Response: Allocate the file if it is not allocated.

GOSC0023E Internal processing error. Contact IBM Support Center

Explanation: The compiler encountered an error when generating method stubs in the .ih file.

User Response: Contact the IBM support center.

GOSC0024E Must specify a name with the \"apply\" modifier.

Explanation: The compiler encountered a missing name in the IDL when generating method stubs in an .ih file.

User Response: Create a name for the IDL modifier and recompile the IDL.

GOSC0025E Cannot find parent of *string* with method *string*.

Explanation: The compiler could not find a parent object in the IDL in the specified search path.

User Response: Include the profile file to specify the correct path.

GOSC0026E Direct-call procedures cannot be overridden: *string*

Explanation: A method attempted to override a static method in the IDL.

User Response: Specify the correct method and recompile the IDL.

GOSC0027E Use `emitter for string` **output.**

Explanation: Internal error.

User Response: Contact the IBM support center.

GOSC0029W Cannot open file \"*string*\".

Explanation: The compiler could not open an IDL or binding file.

User Response: Check if specified file is allocated properly.

GOSC0031E String buffer exceeded [%ld], use -S option.

Explanation: Internal error.

User Response: Contact the IBM support center.

GOSC0033E Cannot continue due to user errors.

Explanation: The compiler cannot continue due to errors described in preceding error messages.

User Response: Correct the preceding errors.

GOSC0034W Program interrupted by user.

Explanation: Program interrupted by the user.

User Response: Restart the program.

GOSC0035E \"*string*\" ambiguous: introduced by \"*string*\" and \"*string*\".

Explanation: The compiler most likely encountered an ambiguous definition of parent methods in the IDL when generating a binding file.

User Response: Check the method definitions in the IDL.

GOSC0036E \"majorversion\" modifier must be between 0 and %ld.

Explanation: A **majorversion** modifier statement is incorrect in the IDL.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0037E \"minorversion\" modifier must be between 0 and %ld.

Explanation: A **minorversion** modifier statement is incorrect in the IDL.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0038E \"filestem\" modifier was not valid. It was either left blank in the IDL, or was longer than 8 characters. Since the filestem represents a PDS member name it cannot exceed 8 characters in length.

Explanation: A **filestem** modifier statement is incorrect in the IDL.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0039E \"functionprefix\" modifier must have a value.

Explanation: A **functionprefix** modifier statement is incorrect in the IDL.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0040E \"externalprefix\" modifier must have a value.

Explanation: A **externalprefix** modifier statement is incorrect in the IDL.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0041E \"externalstem\" modifier must have a value.

Explanation: A **externalstem** modifier statement is incorrect in the IDL.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0042W Cannot use both \"private\" and \"public\" modifiers.

Explanation: The compiler encountered both a **private** and **public** modifier statement, which is not allowed.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0043W \"private\" modifier should not have a value.

Explanation: A value was specified for a **private** modifier statement in the IDL, which is not allowed.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0044W \"public\" modifier should not have a value.

Explanation: A value was specified for a **public** modifier statement in the IDL, which is not allowed.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0045W Cannot use both \"before\" and \"after\" modifiers.

Explanation: The compiler encountered both a **before** and **after** modifier statement, which is not allowed.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0046W \"before\" modifier should not have a value.

Explanation: A value was specified for a **before** modifier statement in the IDL, which is not allowed.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0047W \"after\" modifier should not have a value.

Explanation: A value was specified for an **after** modifier statement in the IDL, which is not allowed.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0048W \"majorversion\" modifier must have a value.

Explanation: A value was not specified for a **major_version** modifier statement in the IDL and a value is required.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0049W \"minorversion\" modifier must have a value.

Explanation: A value was not specified for a **minor_version** modifier statement in the IDL and a value is required.

User Response: Correct the modifier statement and recompile the IDL.

GOSC0050E Input and output files are the same.

Explanation: An input and a binding file had the same name which is not allowed.

User Response: Rename one of the files.

GOSC0051W No \"*\" to remove from a strict IDL class reference.

Explanation: There were no asterisks for the compiler to remove.

User Response: None required.

GOSC0052W In getBaseDesc: No descriptor for type \"*string*\" using '?'.

Explanation: A descriptor was not specified for a type defined in the IDL.

User Response: Specify a descriptor and recompile.

GOSC0053W In setBaseDesc: Already set \"*string*\".

Explanation: A type that was already set was defined in the IDL.

User Response: Correct the type specification and and recompile the IDL.

GOSC0054W Incorrect descriptor format.

Explanation: A valid descriptor was not specified for a type defined in the IDL.

User Response: Specify a valid descriptor and recompile the IDL.

GOSC0056E Cannot allocate %ld bytes.

Explanation: The compiler ran out of storage.

User Response: Increase the region size or correct the erroneous storage allocation. If this action doesn't work, contact the IBM support center.

GOSC0057I *string***: \"***string***\", Version:** *string***.**

Explanation: The compiler is running at the indicated version.

User Response: None required.

GOSC0058I Licensed Materials - Property of IBM5645-001 (C) Copyright IBM Corp. 1992,1997All Rights Reserved.US Government Users Restricted Rights- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Explanation: The compiler has the indicated copyright.

User Response: None required.

GOSC0059I Licensed Materials - Property of IBM.

Explanation: The compiler has the indicated licensing statement.

User Response: None required.

GOSC0060I US Government Users Restricted Rights - Use, duplication or

Explanation: The compiler has the indicated licensing statement.

User Response: None required.

GOSC0061I Disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Explanation: The compiler has the indicated licensing statement.

User Response: None required.

GOSC0062I Date Last Modified: *string string*

Explanation: The compiler was last modified at the indicated date.

User Response: None required.

GOSC0063I Date Last Compiled: *string***]**

Explanation: The IDL was last compiled at the indicated date.

User Response: None required.

GOSC0064E Two source files specified: \"*string*\" and \"*string*\".

Explanation: Two IDL source files were specified.

User Response: Delete the extra -d compiler option and resubmit.

GOSC0065E No source file specified.

Explanation: An input IDL source file was not specified.

User Response: Specify an input IDL file with the -d compiler option and resubmit.

GOSC0066E Unknown entry type: *dstring***.**

Explanation: The type specified for a constant declaration in the IDL was not valid.

User Response: Specify a valid type and recompile the IDL.

GOSC0067W Could not remove file *string*

Explanation: A fatal error occurred and the compiler was not able to remove an incomplete file.

User Response: Remove the indicated file.

GOSC0068I Removed \"*string*\".

Explanation: The compiler removed the indicated file.

User Response: None required.

GOSC0069E Must specify class name.

Explanation: The compiler cannot find a parent class or a class name is blank.

User Response: Indicate the correct class and recompile the IDL.

GOSC0070E In overrideMethod: Trying to override class *string***.**

Explanation: The method being overridden in the IDL is not valid.

User Response: Specify a valid method and recompile.

GOSC0071W Ignoring duplicate identifier \"*string*\".

Explanation: The indicated keyword was specified more than once.

User Response: Correct the duplicate keyword and recompile the IDL.

GOSC0072W \"...\" not valid in data declarations.

Explanation: A '...' was specified in a data declaration.

User Response: Remove the '...' and recompile the IDL.

GOSC0074W *string* **is already used.**

Explanation: The indicated group name was already used in the IDL.

User Response: Use a different group name and recompile the IDL.

GOSC0075E *string* **is already defined.**

Explanation: The indicated class name was already defined in the IDL.

User Response: Fix the class definition and recompile the IDL.

GOSC0076E *string* **is already declared.**

Explanation: The name was already used in a declaration in the IDL.

User Response: Use a unique name and recompile the IDL.

GOSC0077E *string* **is already declared in \"***string***\".**

Explanation: The indicated name was already declared in the indicated method in the IDL.

User Response: Use a unique name and recompile the IDL.

GOSC0079E Redefinition of modifier \"*string***\" for class statement ignored.**

Explanation: The indicated modifier was already defined in the IDL.

User Response: Correct the error and recompile.

GOSC0081W To override the private method \"*string***\", use the \"private\" modifier.**

Explanation: A class definition attempted to override a private method without specifying the private modifier in the IDL.

User Response: Use the **private** modifier and recompile the IDL.

GOSC0082W Internal processing error. Contact IBM Support Center

Explanation: Internal error.

User Response: Contact the IBM support center.

GOSC0083E \"*string***\" modifier clashes with class name \"***string***\",**
add an additional character to \"*string***\".**

Explanation: The indicated modifier is the same as a class name in the IDL.

User Response: Correct the problem and recompile the IDL.

GOSC0086E\" *string***\"has not been declared.**

Explanation: The indicated type for a variable is not valid.

User Response: Correct the variable definition and recompile the IDL.

GOSC0087E \"*string***\" is not a parent.**

Explanation: The indicated class is not a parent class in the IDL.

User Response: Correct the reference to the class and recompile the IDL.

GOSC0089E Parent \"*string***\" repeated.**

Explanation: The indicated parent class already exists in the IDL.

User Response: Correct the definition of the class and recompile the IDL.

GOSC0090E Override method \"*string***\" not in base class.**

Explanation: A class attempted to override an undefined method in the IDL.

User Response: Correct the class definition and recompile the IDL.

GOSC0091E \"*string*\", cannot override \"nooverride\" methods.

Explanation: A class attempted to override a method that was specified to not be overridden by the **nooverride** modifier in the IDL.

User Response: Correct problem as directed.

GOSC0092E \"*string*\", cannot override \"procedure\" methods.

Explanation: The IDL attempted to override a procedure method. A procedure method cannot be overridden.

User Response: Correct the **override** modifier and recompile.

GOSC0093E No prototype for method *string*

Explanation: The IDL tried to use a method that is not defined.

User Response: Correct the method reference or method definition and recompile.

GOSC0094E \"*string*\" has already been used as a class name.

Explanation: A class definition specified a duplicate class name in the IDL.

User Response: Change the class name and recompile the IDL.

GOSC0095E Scope stack is empty.

Explanation: An internal compiler error occurred.

User Response: Contact the IBM support center.

GOSC0096E Scope Stack overruns Heap!

Explanation: An internal compiler error has occurred.

User Response: Contact the IBM support center.

GOSC0097E Cannot find \"*string*\" emitter.

Explanation: The compiler cannot find the specified emitter type because it does not exist.

User Response: Correct the emitter type and recompile.

GOSC0098E Cannot use -u flag with .csc files.

Explanation: The value of the -u compiler option is not valid.

User Response: Use the -h compiler option to find out the supported -u option.

GOSC0099I Return Code *dstring***.**

Explanation: This message indicates the return code from the compiler.

User Response: If necessary, check and correct errors indicated by preceding error messages.

GOSC0100W Unknown option %c

Explanation: The indicated compiler option is unknown.

User Response: Use the -h compiler option to find out the supported option.

GOSC0101E Expecting argument for *string* **option.**

Explanation: The indicated compiler option does not have a valid specified value.

User Response: Use the -h compiler option to find out the supported option.

GOSC0102W Release \"*string*\" has not been declared.

Explanation: A **release_order** modifier on an implementation statement indicated a method that was not declared.

User Response: Correct the **release_order** modifier or declare the method.

GOSC0104W \"*string*\" is not in releaseorder.

Explanation: The indicated method is not in the release order.

User Response: Indicate the method in the release order.

GOSC0107E Cannot load *string*.

Explanation: The indicated emitter is not supported.

User Response: Specify a supported emitter.

GOSC0108E Cannot unload *string*.

Explanation: Internal error.

User Response: Contact the IBM support center.

GOSC0109E line \"*string* ...\" exceeds maximum length of *dstring*.

Explanation: An IDL line exceeded the maximum length of 252 characters.

User Response: Continue the IDL line and recompile the IDL.

GOSC0110E token \"*string* ...\" exceeds maximum length of *dstring*.

Explanation: An IDL token exceeded the maximum length of 252 characters.

User Response: Shorten the IDL token and recompile the IDL.

GOSC0112E Unexpected end of file. No \"endpassthru;\" found.

Explanation: An end for a **passthru** statement in the IDL was not indicated.

User Response: Specify an end for the **passthru** statement and recompile the IDL.

GOSC0113E Newline in modifier string.

Explanation: There was no comma for a continuation or semicolon for the end of a modifier statement in the IDL.

User Response: Specify the necessary comma or semicolon and recompile the IDL.

GOSC0117E Trapped segmentation fault.

Explanation: Internal storage error.

User Response: Contact the IBM support center.

GOSC0118E Empty source file.

Explanation: The specified IDL input file was empty.

User Response: Specify the correct input file.

GOSC0119E *string* around token \"*string*\".

Explanation: Incorrect delimiter in the indicated text in the IDL.

User Response: Correct the delimiter and recompile the IDL.

GOSC0120E Comments should appear after ';' in statements or after ',' in prototypes.

Explanation: Placement of a comment in the IDL is not valid.

User Response: Correct the placement of the comma and recompile the IDL.

GOSC0121E Missing \"class\" section before \"*string*\" section.

Explanation: A class definition is not found before the indicated section in the IDL.

User Response: Define the class or correct the section and recompile the IDL.

GOSC0122E Duplicate or misplaced \"*string*\" statement.

Explanation: The indicated statement is a duplicate or misplaced.

User Response: Correct the statement and recompile the IDL.

GOSC0123W Only one releaseorder per class is permitted.

Explanation: Two **release_order** modifiers were indicated for the same class in the IDL.

User Response: Delete or move one of the **release_order** modifiers and recompile the IDL.

GOSC0124W Cannot specify negative string size for -S. Value *dstring* ignored.

Explanation: A negative value is specified for the -s compiler option.

User Response: Specify a valid value.

GOSC0125W Cannot specify negative string size for -C. Value *dstring* ignored.

Explanation: A negative value is specified for the -C compiler option.

User Response: Specify a valid value.

GOSC0126E Structures require at least one member.

Explanation: A structure is defined without contents in the IDL.

User Response: Correct the structure and recompile the IDL.

GOSC0127E \"*string*\" is not a constant.

Explanation: The indicated data item in the IDL was not a constant as expected.

User Response: Correct the constant reference or definition and recompile the IDL.

GOSC0128E Illegal use of releaseorder as modifier value.

Explanation: A **releaseorder** modifier is misplaced in the IDL.

User Response: Correct the placement of the **release_order** modifier and recompile the IDL.

GOSC0129E Class required for \"metaclass\" modifier.

Explanation: The class value is missing from a **metaclass** modifier or the class specified is not defined or valid in the IDL.

User Response: Correct the **metaclass** modifier or class definition and recompile the IDL.

GOSC0130E Metaclass must be different from class.

Explanation: The class value for a **metaclass** is the same as the metaclass name in the IDL.

User Response: Correct the class name specified and recompile the IDL.

GOSC0131E %lu is out of range %ld - %ld for \"*string*\" operator.

Explanation: The indicated value is not within the required range for the indicated operator in the IDL.

User Response: Correct the value and recompile the IDL.

GOSC0132E \"*string*\" not a valid operand for \"*string*\" operator.

Explanation: The indicated operand is not valid for the indicated operator in the IDL.

User Response: Correct the problem and recompile the IDL.

GOSC0133E IDL does not permit nested structures or unions.

Explanation: IDL does not permit nested structures or unions.

User Response: Correct the structure or union and recompile the IDL.

GOSC0134E \"*string*\" undefined for \"*string*\", size not known.

Explanation: The type in a variable definition in the IDL is not valid.

User Response: Correct the type in the variable definition and recompile the IDL.

GOSC0135E Parameter \"*string*\" repeated.

Explanation: The indicated parameter is a duplicate of a previous parameter.

User Response: Correct the indicated parameter.

GOSC0136E Attributes cannot be explicit arrays.

Explanation: A class attribute was defined as an explicit array.

User Response: Correct the attribute definition.

GOSC0137E Constant expression type mismatch for \"*string*\".

Explanation: A class attribute was defined with a type that is not valid.

User Response: Define the attribute with a valid type.

GOSC0138E Cannot apply `'%c'` to string \"*string*\".

Explanation: A string was applied to a unary operation.

User Response: Correct the operation.

GOSC0139E \"*string*\" is undefined for floating point operands.

Explanation: A string in the IDL was applied to a unary operation.

User Response: Correct the problem and recompile the IDL.

GOSC0140E Duplicate case label for value \"*string*\".

Explanation: An operation in the IDL contained an undefined operator.

User Response: Use a defined operator in the operation and recompile the IDL.

GOSC0141E Only one default statement permitted per case statement.

Explanation: A case statement in the IDL had more than one default.

User Response: Specify a single default for the case statement and recompile the IDL.

GOSC0142E implementation already declared for \"*string*\".

Explanation: The indicated class in the IDL had more than one implementation statement.

User Response: Specify a single implementation statement for the class and recompile the IDL.

GOSC0143E \"unsigned *string*\", only \"short\" and \"long\" types can be unsigned.

Explanation: A variable defined as unsigned in the IDL was not defined as a **short** or **long** type.

User Response: Correct the variable definition and recompile the IDL.

GOSC0144E \"*string*\", is not an exception.

Explanation: A **raises** expression in the IDL specified an exception that is not valid.

User Response: Correct the raise expression and recompile.

GOSC0146W Metaclass \"*string*\" is not descended from \" SOMClass\".

Explanation: The indicated metaclass is not descended from SOMClass.

User Response: Correct the metaclass definition and recompile the IDL.

GOSC0147W Ignoring repeated Metaclass modifier: \"*string*\".

Explanation: A metaclass definition had a duplicate **metaclass** modifier.

User Response: Remove the duplicate **metaclass** modifier and recompile the IDL.

GOSC0148W Non-portable \"int\" will be generated in emitter files.

Explanation: The indicated integer is not defined with a SOM type code.

User Response: Correct the integer definition and recompile the IDL.

GOSC0149E \"*string*\" is negative.

Explanation: An unsigned constant was defined with a negative value.

User Response: Correct the constant definition. and recompile the IDL.

GOSC0150E Exception expects {}.

Explanation: A brace was missing in an exception declaration in the IDL.

User Response: Correct the exception declaration and recompile the IDL.

GOSC0151E Mixed type expressions are illegal.

Explanation: The data in an expression had different types.

User Response: Correct the mixed data types in the expression

GOSC0152E \"*string*\" undefined for string constants.

Explanation: One operand in the expression might not be a string.

User Response: Check whether both expressions are strings.

GOSC0153E Case expression \"*string*\" must be an integral constant.

Explanation: The indicated expression did not evaluate to an integral constant.

User Response: Check that the expression is an integral constant.

GOSC0154E Case expression \"*string*\" must match union discriminator \"*string*\".

Explanation: The indicated case expression and union discriminator do not match. The case expression must match the defined type of the union discriminator.

User Response: Correct the case expression.

GOSC0155E Case expression \"*string*\" exceeds size of union discriminator \"*string*\".

Explanation: A case expression length is greater than the maximum length for the union discriminator type. For example, the union discriminator type is **char** and the case expression integer value is greater than 256.

User Response: Evaluate and correct the union structure.

GOSC0156E Non-standard IDL use of explicit *'s.

Explanation: The IDL contained an asterisk.

User Response: Remove the asterisk from the IDL.

GOSC0157E Non-standard IDL \"implementation\" section.

Explanation: An implementation statement is not CORBA compliant.

User Response: Either remove the -mcorba compiler option or add the #ifdef __SOMIDL__ directive.

GOSC0158E Non-standard IDL unsigned \"*string*\", should be \"short\" or \"long\".

Explanation: An integer was not defined as either a long or short type in the IDL.

User Response: Correct the definition and recompile the IDL.

GOSC0159I Running preprocessor against *string*

Explanation: The compiler is running the preprocessor against the IDL input file.

User Response: None required.

GOSC0160I Number of errors: *dstring*

Explanation: This message indicates the number of errors encountered during the compile.

User Response: Correct the errors.

GOSC0161I usage: *string* **[-D :I:V:c:d:h:i:m:prvw] filename**

Explanation: This message provides information about how each compiler option was specified.

-C <n> - size of comment buffer (default: *dstring*)
-D <DEFINE> - same as DEF() option for cpp.
-E <var>=<value> - set environment variable.
-I <INCLUDE> - same as SE() option for cpp.
-S <n> - size of string buffer (default: *dstring*)
-U <UNDEFINE> - same as -U option for cpp.
-V - show version number of compiler.
-c - ignore all comments.
-d <path> - Path or PDS stem for each emitted file.
-h this message.
-i <file> - use this file name as supplied.
-m<name[=value]>
add global modifier.
-p shorthand for -D__PRIVATE__.
-r check releaseorder entries exist (default: FALSE).
-s <string> - replace SMEMIT variable with <string>
-u update interface repository.

-v verbose debugging mode (default: FALSE).
-w don't display warnings (default: FALSE).

Modifiers:

addprefixes
adds 'functionprefix' to method names in template file
[no]addstar
[no]add '*' to C bindings for interface references.
corba check the source for CORBA compliance.
csc force running of OIDL compiler.
emitappend
append the emitted files at the end of 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>
specify a local pre-processor to use.
tcconsts generate CORBA TypeCode constants.

Note: All command-line modifiers can be set as an environment variable by changing them to UPPERCASE and pre-appending " | \"SM\" to them.|

Environment Variables:

SMEMIT [h;ih;c;xh;xih;xc]\n :.br;emitters to run (default : h;ih).
SMINCLUDE
<dir1>[%c<dir2>[;...]]\n (where to search for #include IDL files)
SMKNOWNEXTS
ext[;ext]+\n (add headers to user written emitters)
SMTMP <dir>\n (directory to hold intermediate files)
SOMIR <path>[;<path>]+\n (list of IRs to search). Define or export the SMOE environment variable to change the default file system from MVS to the HFS.

Pragmas:

#pragma somemittypes on - turn on emission of global types.
#pragma somemittypes off - turn off emission of global types.
#pragma modifier <modifier stm>; - instead of modifier statement.

User Response: None required.

GOSC0208E Attribute name \"*string*\" too long.

Explanation: An attribute name that you specified in the IDL exceeded the the maximum length of 250 characters.

User Response: Shorten the attribute name and recompile the IDL.

GOSC0210E Only 1 declarator permitted per SOMFOREIGN typedef: \"*string*\"

Explanation: There are two definitions of the SOMFOREIGN typedef in the IDL.

User Response: Remove one of the definitions.

GOSC0211E Missing \"impctx\" modifier for foreign type \"*string*\"

Explanation: Implementation context information is missing.

User Response: Specify the information with the **impctx** modifier.

GOSC0212E Implementation section for interface \"*string*\" requires align=%hd for correct placement of instance data

Explanation: Alignment information was not specified when it was required.

User Response: Provide alignment information.

GOSC0213E Could not resolve type \"*string*\"

Explanation: When you ran the IR emitter, the indicated type for a variable is not valid.

User Response: Correct the variable definition and recompile the IDL.

GOSC0214E Unexpected entry *string* encountered

Explanation: An internal compiler error occurred.

User Response: Contact the IBM support center.

GOSC0215E Could not resolve type for member \"*string*\"

Explanation: A union type declaration contained an undefined type.

User Response: Correct the type code.

GOSC0216E %ld type-related errors

Explanation: This message indicates the total number of errors during IR processing.

User Response: Correct the errors.

GOSC0217E Cannot open Interface Repository file \"*string*\" (*string*)

Explanation: The compiler cannot open the indicated IR file.

User Response: Check allocation and RACF authorization of the file.

GOSC0218E Cannot open Interface Repository

Explanation: The compiler cannot open the IR file.

User Response: Check allocation and RACF authorization of the file.

GOSC0219E Ambiguous use of \">>\", separate with space (\"> >\").

Explanation: A space is missing in a >> expression in the IDL.

User Response: Correct the expression.

GOSC0220E filestems \"*string*\" and \"*string*\" differ.

Explanation: All **filestem** modifiers in the IDL must specify the same file name.

User Response: Correct the **filestem** modifier and recompile the IDL.

GOSC0221I qualifying names \"*string*\" -> \"*string*\".

Explanation: This message indicates a qualified name.

User Response: None required.

GOSC0222I unqualifying names \"*string*\" -> \"*string*\".

Explanation: This message indicated an unqualified name.

User Response: Correct errors as directed.

GOSC0223E *'s are not permitted in sequence declarations.

Explanation: A pointer was specified in a sequence declaration.

User Response: Correct the sequence declaration.

GOSC0224E *'s are not permitted in string declarations.

Explanation: A pointer was specified in a string declaration.

User Response: Correct the string declaration.

GOSC0225E \"*string***\" is a pointer type, unions expect an integer type.**

Explanation: A pointer was specified in a union declaration.

User Response: Correct the union declaration.

GOSC0226E \"*string***\" requires parameter before final \"***string***\".**

Explanation: A parent variable argument list does not exist when a child class has variable arguments.

User Response: Correct the parent or child class.

GOSC0227I Running emitter: *string*

Explanation: The specified emitter is running.

User Response: None required.

GOSC0228I Emitter completed: *string* **Output Dataset=***string*

Explanation: The processing for the specified emitter has completed.

User Response: None required.

GOSC0231E The \"*string***\" modifier is a user defined modifier.**

Explanation: The indicated modifier is a modifier defined by MVS.

User Response: Correct error.

GOSC0232I An error occurred opening file \"*string***\" for mode \"***string***\".**

User Response: Check that the file is allocated and has the correct RACF authorization.

GOSC0233I The preprocessor failed. (RC=%i)

Explanation: The compiler preprocessor failed.

User Response: Check and correct errors indicated by preceding error messages.

GOSC0235E Only SOMObject's initializers may be overridden: *string*

Explanation: The IDL attempted to override one of the default initializer methods. An override of the initializer method is not allowed.

User Response: Correct the IDL and recompile.

GOSC0236E Reintroduction of *string* **as** *string*
*string***::***string* **would hide static method**
*string***::***string*

Explanation: A method originally introduced by one class (for example, X::foo) has been reintroduced by a descendant class (as, for example, Y::foo), which is not allowed.

User Response: The descendant class should override the original method.

GOSC0239W public, protected, or static data \\"*string***\\" is not in releaseorder.**

Explanation: A public, protected, or static variable defined in the implementation statement for a class was not specified in the **release_order** modifier.

User Response: Include the variable with the **release_order** modifier.

GOSC0240E Method *string* **is not inherited into** *string***, so it cannot be migrated.**

Explanation: Conflict of old and new IDL.

User Response: Check IDL inheritance structure.

GOSC0241W The *string* **emitter doesn't support OIDL classes. Use ctoi to convert OIDL to IDL.**

Explanation: ctoi converter not available in MVS.

User Response: None.

GOSC0243E *string***: oneway methods cannot raise exceptions.**

Explanation: A oneway method can not raise exception condition.

User Response: Correct idl and recompile.

GOSC0244E *string***: Noself and noenv modifiers can only be used on direct-call procedures.**

Explanation: noself or noenv used with a non-direct call procedure.

User Response: Correct idl and recompile.

GOSC0245E *string***: Oneway methods cannot return results or accept out parameter.**

Explanation: oneway methods cannot return results.

User Response: Correct idl and recompile.

GOSC0246E *string***: Cannot use select to override an initializer.**

Explanation: Select cannot be used with initializer.

User Response: Correct idl and recompile.

GOSC0247E *string***: select must indicate an immediate parent.**

Explanation: Select modifier has to specify a parent.

User Response: Correct idl and recompile.

GOSC0248E *string***: control argument missing from initializer.**

Explanation: somInitCtrl definition must be in procedure declaration.

User Response: Correct idl and recompile.

GOSC0249E *string***: additional arguments are required for user-defined initializers.**

Explanation: There has to be one more argument in addition to SomInitCtrl argument in the procedure definition.

User Response: Correct idl and recompile.

GOSC0250E *string***: illegal releaseorder entry.**

Explanation: Only method can be specified after releaseorder.

User Response: Correct idl and recompile.

GOSC0252E *string***: initstyle=fast requires ABI 3.**

Explanation: Use of initstyle=fast has to specify ABI 3.

User Response: Correct idl and recompile.

GOSC0253E *string***: initstyle=fast requires overrides of somDefaultInit, somDestruct, and a copy method.**

Explanation: Use of initstyle=fast has to override somDefaultInit, somDestruct and a copy method.

User Response: Correct idl and recompile.

GOSC0254W *string***: not an initializer.**

Explanation: Original method does not have an initializer.

User Response: Correct idl and recompile.

GOSC0255W *string***: initializers must return void.**

Explanation: Initialized Procedure definition cannot return any value other than void.

User Response: Correct idl and recompile.

GOSC0256W Open failed for IR file: *string*

Explanation: Cannot open IR file when creating IR Index file.

User Response: Check if specified file is allocated properly.

GOSC0257W Read failed for IR file: *string*

Explanation: Cannot open IR file when creating IR Index file.

User Response: Check if specified file is allocated properly.

GOSC0258W Not authorized to create index for IR file: *string*

Explanation: Cannot open IR file due to READ Access authority.

User Response: Obtain Read authority of IR file.

GOSC0259W Index creation failed for IR file: *string*

Explanation: Create of Index file failed.

User Response: Check if there is any other accompanying message.

GOSC0260W Please delete file: *string*

Explanation: Delete specified Index file.

User Response: Delete specified Index file as requested.

GOSC0261W Environment variable SOMIR not set

Explanation: Environment SOMIR=ir data set name required.

User Response: Correct SOMENV file.

GOSC0264W Index does not exist for IR file: *string*

Explanation: Cannot locate Index file.

User Response: Determine if Index file exists.

GOSC0265W IR file has index: \"*string***\"**

Explanation: IR file has no Index.

User Response: Create Index file for the IR.

GOSC0266W
GOSC0267W
GOSC0268W
GOSC0269W
GOSC0270W
GOSC0271W
GOSC0272W
GOSC0273W
GOSC0274W
GOSC0275W
GOSC0276W
GOSC0277W

Explanation: somdeps [options] file where options are:

\t-Dname[=def]	Defines name as def (default is 1).
\t-Idir	Add directory dir to #include file search path.
\t-Uname	Removes definition of name.
\t-g	Lists dependency files found in -I directories
\t	(default lists dependency files only found in current directory).
\t-h	Usage message.
\t-ofile	Put output in file (default is stdout).
\t-sext	Specifies ext as the dependency file extension
\t	for idl file (default is ih).

GOSC0278W C++ bindings cannot be produced without stars.

Explanation: To create C++ binding stars is a must.

User Response: Add -maddstar to SOM compiler command line.

GOSC0279W *string***: Cannot override both somInit and somDefaultInit**

Explanation: Can only override either somInit or somDefaultInit.

User Response: Correct idl and recompile.

GOSC0280W Modifier \"*string***\" specified after first use of \"***string***\".**

Explanation: The `'Enum'` has already been referenced (e.g. by a `'Union'`), so it is too late to change the origin now.

User Response: Correct idl and recompile.

GOSC0281E An internal routine attempted to display an invalid message.

Explanation: Internal error.

User Response: Contact the IBM support center.

GOSC0283E Character literal *string* **contains more than one character**

Explanation: A character literal can only be one character.

User Response: Correct error.

GOSC0284E Character literal *string* **contains more than characters than the sizeofan integer. The right most characters are retained**

Explanation: The literal contained more characters than the integer size that was defined and implemented.

User Response: Check that the literal contains the correct number of characters.

GOSC0285E '/*' detected in C comment

Explanation: A comment exists inside a comment.

User Response: Correct error.

GOSC0286I *string* **condition evaluates to** *string*

Explanation: Condition expression evaluates to the indicated value.

User Response: None required.

GOSC0287E Double byte character sequence must contain an even number of bytes

Explanation: Double byte character sequence contained an odd number of bytes.

User Response: Correct error.

GOSC0288I Identifier *string* **assigned default value of 0 on #***string* **directive**

Explanation: A value was not explicitly defined so it was set to zero.

User Response: Evaluate and correct if necessary.

GOSC0289I defined(*string***) evaluates to** *string*

Explanation: The indicated identifier evaluates to the indicated value.

User Response: None required.

GOSC0290I #*string* **directive has been ignored**

Explanation: The compiler preprocessor ignored the indicated directive.

User Response: Evaluate and correct if necessary.

GOSC0291E Division by zero on #*string* **directive**

Explanation: The divisor evaluated to zero for the indicated expression.

User Response: Evaluate and correct if necessary.

GOSC0293E The parameter name *string* **has already been used for the macro** *string*

Explanation: A duplicate parameter was encountered.

User Response: Correct error.

GOSC0294E #elif encountered with no matching #if

Explanation: A #if did not precede a #elif statement.

User Response: Evaluate and correct if necessary.

GOSC0295E #else encountered with no matching #if

Explanation: A #if did not precede a #else statement.

User Response: Correct error.

GOSC0296E Empty source file

Explanation: Preprocessor received an empty IDL file.

User Response: Correct error.

GOSC0297E Empty argument specified for parameter *string* **of the macro** *string*

Explanation: Macro did not contain at least one argument.

User Response: Specify the arguments for a macro.

GOSC0299E #endif encountered with no matching #if

Explanation: A #if directive did not precede a #endif statement.

User Response: Correct the #if directive.

GOSC0300E Expecting newline character before end of file

Explanation: A directive was not completed before the end of the file.

User Response: Complete the directive.

GOSC0301E Escape sequence *string* **is out of the range 0-***string*

Explanation: The hexadecimal value to represent a character in the escape sequence was out of range.

User Response: Correct error.

GOSC0302E Missing #endif directive

Explanation: The number of #if and #endif directives do not match in the IDL.

User Response: Correct error.

GOSC0303E Expecting macro name on #*string* **directive but found** *string*

Explanation: A macro name was not valid or not specified on a directive.

User Response: Correct the directive specification.

GOSC0304E Expecting end of line on #*string* **directive but found** *string*

Explanation: Incorrect data or more data than expected was specified for the directive.

User Response: Correct error.

GOSC0305E Too many arguments specified for the macro *string***. Extra arguments ignored**

Explanation: More arguments than expected were specified for the macro.

User Response: Correct error.

GOSC0306E Comment which began on line *string* did not end before the end of file

Explanation: Comment did not contain an end delimiter.

User Response: Include comment end delimiter.

GOSC0266E Continuation sequence found at end of file. It will be ignored

Explanation: Comment did not contain an end delimiter.

User Response: Correct error.

GOSC0307E Unable to open file *string*. Reason: *string*

Explanation: The compiler preprocessor cannot open the indicated file.

User Response: Check the allocation and RACF authorization for the file.

GOSC0308E Unable to read file *string*. Reason: *string*

Explanation: The compiler preprocessor cannot read the indicated file.

User Response: Check the allocation and RACF authorization for the file.

GOSC0309E The floating point literal *string* is out of range

Explanation: The indicated floating point literal is out of the required range.

User Response: Define the literal as a double precision floating point.

GOSC0310E The name *string* may not be defined as a macro

Explanation: A macro was defined with a reserved name.

User Response: Define the macro with a different name.

GOSC0311E The name *string* may not be undefined as a macro

Explanation: A macro was undefined with a reserved name.

User Response: Undefine the macro with a different name.

GOSC0312E Empty header encountered on #include directive

Explanation: The file specified with the #include directive is empty.

User Response: Correct error.

GOSC0313E Illegal character *string*

Explanation: The indicated character is not allowed.

User Response: Correct error.

GOSC0314E Illegal use of the token pasting operator on the macro *string*

Explanation: The source IDL contains ## as the first or last token of a macro definition or consecutive ## tokens in a macro definition.

User Response: Correct the error and recompile the IDL.

GOSC0315E Syntax error in constant expression on #*string* directive. Found *string*

Explanation: The preprocessor found data that is not a number.

User Response: Correct error.

GOSC0316E Illegal escape sequence *string***. The \\ is ignored**

Explanation: The preprocessor encountered an undefined escape sequence and did not apply it.

User Response: Correct error.

GOSC0317E Illegal floating point literal *string*

Explanation: The definition contains data that is not a number or point.

User Response: Correct error.

GOSC0318E Illegal hexadecimal literal *string*

Explanation: The literal contains a character other than 0-F.

User Response: Correct error.

GOSC0319E Illegal hexadecimal escape sequence *string***. The \\ is ignored**

Explanation: The preprocessor encountered an undefined escape sequence and did not apply it.

User Response: Correct error.

GOSC0320E Illegal integer literal *string*

Explanation: The definition contains data that is not a number.

User Response: Correct error.

GOSC0321E Illegal parameter list for the macro *string* **but found** *string*

Explanation: The use of the macro parameter is not the same as defined.

User Response: Correct error.

GOSC0322E Illegal wchar_t character. Codepoint *string*

Explanation: A double byte character does not have a hexadecimal code point value associated with it.

User Response: Correct error.

GOSC0323E Illegal multibyte character *string* **encountered**

Explanation: The indicated character is not allowed.

User Response: Correct error.

GOSC0324E Illegal octal literal *string*

Explanation: The literal contains a character other than 0-7.

User Response: Correct error.

GOSC0325E Non octal digit *string* **encountered on octal literal**

Explanation: The literal contains a character other than 0-7.

User Response: Correct error.

GOSC0326E The file *string* **has already been #included**

Explanation: The preprocessor encountered a duplicate #include directive indicating the same file.

User Response: Correct error.

GOSC0327E Expecting header name on #include directive but found *string*

Explanation: The preprocessor encountered a #include directive that did not specify a file

User Response: Correct the #include directive.

GOSC0328I #include search found file *string*

Explanation: The preprocessor found the file that the #include directive specified.

User Response: None required.

GOSC0329E Cannot #include file *string***. Maximum #include nesting of** *string* **has been reached**

Explanation: The preprocessor could not include a file because the number of nested #include directives exceeded the maximum. The maximum for #include directives is the maximum for a short integer type on your system.

User Response: Correct error.

GOSC0330E Wide character string *string* **not allowed on #include directive**

Explanation: The source IDL code specified an #include preprocessor directive to automatically include another file (for example, header files or SOM binding files in the source. The name in the #include statement specified the wide character literal shown in the message text. The wide character literal is not allowed in the #include directive.

User Response: Remove the wide character literal from the #include directive.

GOSC0331E #include file *string* **not found**

Explanation: The source IDL code specified an #include preprocessor directive to automatically include bindings in the source. The compiler preprocessor could not find the header file.

User Response: The SMINCLUDE environment variable or the -I option on the command line does not include the necessary files.

GOSC0332I #include search attempting file *string*

Explanation: The source IDL code specified an #include preprocessor directive to automatically include bindings in the source. The preprocessor is searching for the header file.

User Response: None required.

GOSC0333E Incomplete argument specified for parameter *string* **of macro** *string*

Explanation: The source IDL code called a macro, for example, to manipulate a class or object. The compiler preprocessor found a parameter value that is not complete.

User Response: Ensure that the macro call specifies parameter values that are valid. If necessary, see *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for the macro syntax.

GOSC0334E Incomplete parameter list specified for macro *string*

Explanation: The source IDL code called a macro, for example, to manipulate a class or object. The compiler preprocessor did not find a complete parameter list on the call. The compiler assumes that the unspecified parameters are empty.

User Response: Ensure that the macro call specifies all required parameters. If necessary, see *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for the macro syntax.

GOSC0335E Preprocessor internal error. File *string***: Line** *string*

Explanation: An internal error occurred.

User Response: Contact the IBM Support Center. Provide the file name and line number shown in the message text.

GOSC0336E Integer literal *string* is out of range

Explanation: The source IDL code processed the integer literal shown in the message text. The integer literal contained a value that is out of range.

User Response: Change the integer literal to specify a value that is within the valid range.

GOSC0337E Wide character literal *string* contains more than one character. The last character will be used

Explanation: The source IDL code processed the wide character literal shown in the message text. The literal contained more than one character, which is not valid. The compiler preprocessor will use the last character in the literal.

User Response: If necessary, change the wide character literal to specify one character with a preceding **L**.

GOSC0338I Line number changed to *string* and file name changed to *string* due to #line directive

Explanation: The compiler preprocessor processed the #line directive, which sets the next source line to the specified line number. As a result, the line number and file name are changed.

User Response: None required.

GOSC0339I Line number changed to *string* due to #line directive "

Explanation: The preprocessor processed the #line directive, which sets the next source line to the specified line number.

User Response: None required.

GOSC0340E Line number *string* on #line directive must contain only decimal digits

Explanation: The line number of the next source line, specified on the #line directive, is the indicated string which is not valid.

User Response: Change the value on the #line directive to a number.

GOSC0341E Expecting file name or end of line on #line directive but found *string*

Explanation: The #line directive contains a valid line number but a file name that is not valid.

User Response: Change the value on the #line directive to one that is within the valid range.

GOSC0342E Expecting line number on #line directive but found *string*

Explanation: The line number specified on the #line directive is the indicated string which is not valid.

User Response: Change the value on the #line directive to integer number.

GOSC0343E Wide string literal *string* not allowed on #line directive

Explanation: The line number specified on the #line directive is the indicated string literal which is not valid.

User Response: Change the value on the #line directive to an integer number.

GOSC0344E #line value *string* must not be zero

Explanation: The line number specified on the #line directive, is zero, which is not valid.

User Response: Change the value on the #line directive to one that is within the valid range.

GOSC0345E #line value *string* is out of range

Explanation: The line number specified on the #line directive is not within the valid range.

User Response: Change the value on the #line directive to one that is within the valid range.

GOSC0346E Illegal macro parameter list for macro *string***. Expecting an identifier but found** *string*

Explanation: On a call of the indicated macro, the compiler found the indicated string rather than a valid identifier.

User Response: Ensure that the macro call specifies a valid identifier. If necessary, see *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for the macro syntax.

GOSC0347E Macro name *string* **is also a keyword**

Explanation: The name for the macro was a reserved keyword.

User Response: Ensure that the macro call specifies a valid macro name. If necessary, see *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for the macro syntax.

GOSC0349E Some text ignored due to source margins

Explanation: The compiler did not process some of the text in the source file because it extended beyond the source margins.

User Response: Ensure that the text in the source file does not extend beyond the source margins. If necessary, use a backslash character (\) to indicate that the line of text is to flow onto the next line.

GOSC0350E Macro name *string* **is already defined with the same definition**

Explanation: The source IDL code specified the #define preprocessor directive to define a macro with the same name and same definition as a previously defined macro.

User Response: Change the macro definition to specify a different macro name.

GOSC0351I Macro name *string* **was defined in** *string* **on line** *string*

Explanation: This is an informational message that is issued after a message about an invalid macro definition.

User Response: Correct the macro definition.

GOSC0352I #*string* **nesting level is** *string*

Explanation: This message indicates the current level of nesting.

User Response: None required.

GOSC0353E Macro name *string* **is already defined with a different definition**

Explanation: The source IDL code specified the #define preprocessor directive to define a macro with the same name and a different definition than a previously defined macro.

User Response: Change the macro definition to specify a different macro name.

GOSC0354E Out of memory at file *string***, line** *string*

Explanation: The preprocessor of the compiler ran out of memory at the line and file indicated in the message text.

User Response: In the JCL for the job, specify a larger region size on the REGION parameter.

GOSC0355E Pascal string too long. Truncated to *string* **data bytes**

Explanation: The compiler could not process the string because it exceeded the maximum allowable length. The compiler truncated the string to the allowable length.

User Response: None required. If necessary, reduce the size of pascal strings so they are less than or equal to the allowable length.

GOSC0364I Macro name *string* is reserved but the definition will be honored

Explanation: The source IDL code specified the #define preprocessor directive to define a macro. The macro definition was reserved. The compiler will use the definition in the source IDL code.

User Response: Evaluate and change if necessary.

GOSC0365I Macro name *string* is reserved. The undefinition will be honored

Explanation: The source IDL code specified the #undef preprocessor directive to "undefine" a reserved processor definition, which is shown in the message text.

User Response: Evaluate and change as necessary.

GOSC0366E #error text: *string*

Explanation: The preprocessor passed a portion of your program when it should not have and the message indicates the #error directive text.

User Response: Correct your program logic or the compiler parameter list.

GOSC0367E Some text ignored due to sequence columns

Explanation: The IDL file has sequence numbers in the wrong column.

User Response: Fix the sequence numbers in the data set.

GOSC0368E Token skipping due to conditional compilation begins

Explanation: The preprocessor of the compiler is not processing certain tokens because of conditional statements specified on the compile command.

User Response: Evaluate and change as necessary.

GOSC0369E Parameter name expected after stringizing operator on macro *string* but found *string*

Explanation: The preprocessor of the compiler expected the parameter value, shown in the message text, after the macro name, but found a different parameter value, also shown in the message text.

User Response: Ensure that the macro call specifies the correct parameter values. If necessary, see *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for the macro syntax.

GOSC0370E Too few arguments specified for macro *string*. Empty arguments assumed "

Explanation: The indicated macro specified a number of parameters that was less than the required. The preprocessor of the compiler assumes that the unspecified parameters are empty.

User Response: Ensure that the macro call specifies the required number of parameters. If necessary, see *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for the macro syntax.

GOSC0371I Trigraph *string* expanded in a character literal

Explanation: The source IDL code specified a trigraph sequence to represent a single character literal.

User Response: None required.

GOSC0372I Trigraph *string* expanded in a string literal

Explanation: The source IDL code specified a trigraph sequence to represent a string literal.

User Response: None required.

GOSC0373I Undefining macro *string*

Explanation: The source IDL code specified the #undef preprocessor directive to “undefine” a previous processor definition.

User Response: None required.

GOSC0374E Unknown preprocessing directive *string* **ignored**

Explanation: The preprocessor found a directive that is not valid. The name of the directive is specified in the message text.

User Response: Ensure that the directive is spelled correctly and has the correct syntax. If necessary, see the section on preprocessing in C or C++ documentation for information about valid directives.

GOSC0375E Undefining unknown macro name

Explanation: The source IDL code specified an #undefine directive for an unknown macro name.

User Response: Ensure that the macro name appears correctly in the directive statement. If necessary, see *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for the macro syntax.

GOSC0376E Character literal must end before the end of the source line

Explanation: The source IDL code specified a literal that continued to the next source line without indicating a continuation with a backslash and double quotes.

User Response: Ensure that the source IDL specifies a matching double quotation mark at the end of each string constant.

GOSC0377E #include header must end before the end of the source line

Explanation: The source IDL code specified an #include header to automatically include files in the source. The header did not end before the end of the source line in the code, probably because it did not specify an ending **>** symbol.

User Response: Ensure that the #include header specifies a beginning **<** and ending **>** symbol around the header name.

GOSC0378E Character literal on preprocessing directive or false preprocessor conditional block must end before the end of the source line

Explanation: The character literal on a preprocessing directive or false conditional block must appear before any other IDL. Other IDL appeared before the character literal.

User Response: Correct the error and recompile the IDL.

GOSC0379E String literal on preprocessing directive or false preprocessor conditional block must end before the end of the source line

Explanation: The string literal on a preprocessing directive or false conditional block must appear before any other IDL. Other IDL appeared before the character literal.

User Response: Correct the error and recompile the IDL.

GOSC0380E String literal must end before the end of the source line

Explanation: Your program specified a literal string that did not end before the end of a line in the source code and did not indicate continuation by using a backslash and double quotes.

User Response: Ensure that, in your source code, you end all literal strings before the column that indicates the end of a source code line or indicate continuation.

GOSC0395W String literal must end before the end of file

Explanation: The string was missing the ending '"'.

User Response: Locate string and correct error.

GOSC0396W Invalid suboption *string* for *string*. Suboption ignored.

Explanation: This message can occur when the there is an error in the file name specified on the SMINCLUDE environment variable or the -I flag.

User Response: Check the syntax of the names specified on the SMINCLUDE environment variable and any -I flags. Examples of valid MVS PDS stems are: //'SOMMVS.V1R2M0.SGOSIDL' //myown toys Examples of valid HFS path names are: /u/sombear/ ./objects test/ toys By default, names not containing a "/" are considered MVS PDS stems. Use the SMOE environment variable or the -moe modifier if the name should be treated as an HFS path.

GOSC0398E An internal routine attempted to display an invalid message.

Explanation: An internal error occurred.

User Response: Contact the IBM Support Center. _H_SOMCENUS

Distributed SOMobjects (DSOM) Messages

Following are the texts, explanations, and user responses for the messages issued by distributed SOMobjects.

These messages appear in the program listing.

GOSD0006I Sockets class............ *string*

Explanation: This message displays the name of the SOM Sockets subclass that implements the sockets services.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0007I Object reference file..... *string*

Explanation: This message displays the full pathname of the file used to store reference data associated with object references created by this server.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0008I Object reference backup... *string*

Explanation: This message displays the full pathname of the backup file used to mirror the primary reference data file for this server.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0009I Host name................. *string*

Explanation: This message displays the name of the machine on which the server program code is stored.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0010I

```
Implementation alias                Associated classes
=================================================================
```

Explanation: This message displays the implementation aliases and their classes.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0011I

```
Implementation alias                Implementation ID
=================================================================
```

Explanation: This message displays the title of the list of implementation aliases and their implementation id.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0019I Enter regimpl -? for help

Explanation: Enter regimpl -? for help.

User Response: None required.

GOSD0020I

```
To enter interactive mode:
      regimpl
Add implementation:
      regimpl -A -i <str> [-v <str>]
              [-z <str>]
              [-e <str> {<param> ...}] [-s {on|off}]
              [-t <str> {<param> ...} [-t ...]]
Update implementation:
      regimpl -U -i <str> [-v <str>]
              [-e <str> {<param> ...}] [-s {on|off}]
              [-t <str> {<param> ...} [-t ...]]
Delete implementation:
      regimpl -D -i <str> [-i ...]
List implementation(s):
      regimpl -L [-i <str> [-i ...]]
List aliases:
      regimpl -S
Add class(es):
      regimpl -a -c <str> [-c ...] -i <str> [-i ...]
Delete class(es):
      regimpl -d -c <str> [-c ...] [-i <str> [-i ...]]
List classes associated with implementation(s):
      regimpl -l [-i <str> [-i ...]]
where:
  -i <str>    = Implementation alias name
  -v <str>    = Server-class name  (default: SOMDServer)
  -z <str>    = Implementation ID
  -c <str>    = Class name
  -e <str>    = ImplDef Class name (default: ImplementationDef)
     <param>  = Values for additional attributes needed by subclass of
                ImplementationDef
  -s {on|off} = Enable secure server (optional)
  -t <str>    = Transport protocol
     <param>  = Option can be zero or more strings, delimited by
                spaces.
```

Explanation: REGIMPL help text.

User Response: None required.

GOSD0063E Error occurred when adding server to Implementation Repository.

Explanation: The system found an error when adding a server implementation.

User Response: Use the REGIMPL utility to correct the server implementation. For information about how to use REGIMPL to register a server, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0064E Initialization Error: Cannot find Interface Repository.

Explanation: Unable to initialize the DSOM environment because system cannot find the Interface Repository specified in the configuration file.

User Response: Check that all Interface Repositories pointed to by SOMIR environment variable are valid. This can be done by using the IRDUMP utility. If the IRs are okay, contact next level of support.

GOSD0065E Error occurred when updating the Implementation Repository.

Explanation: The system found an error when updating a server implementation.

User Response: Use the REGIMPL utility to correct the server implementation. For information about how to use REGIMPL to register a server, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0069I Class '*string*' is now associated with server '*string*'

Explanation: The specified class is now associated with the specified implementation.

User Response: None required.

GOSD0072I Do you wish to abort this operation? [y/n] :

Explanation: REGIMPL has detected an action that indicates that the user may want to abort the operation, such as not specifying an alias name on an add alias request.

User Response: Enter `'y'` if the current request is to be cancelled, otherwise enter `'n'`.

GOSD0074I Enter the alias of the server that implements the class :

Explanation: REGIMPL is prompting the user to enter the alias name of the server to implement the specified class.

User Response: Enter the alias name of the desired server

GOSD0075I Enter the alias of the server to delete :

Explanation: REGIMPL is prompting the user to enter the alias name of the server to delete in the implementation repository.

User Response: Enter the alias name of the desired server to delete.

GOSD0076I Enter the alias of the server to show :

Explanation: REGIMPL is prompting the user to enter the alias name of the server the user wanted to see information displayed

User Response: ENter the alias name of the desired server

GOSD0077I Enter the alias of the server to update :

Explanation: REGIMPL is prompting the user to enter the alias name of the server to update in the implementation repository.

User Response: Enter the alias name of the desired server to update.

GOSD0078I Enter an alias for a new server :

Explanation: REGIMPL is prompting the user to enter the alias name of the server to add in the implementation repository.

User Response: Enter the alias name of the desired server to add.

GOSD0080I Enter the name of the class to delete :

Explanation: REGIMPL is prompting the user to enter the class name of the class to delete from the naming service

User Response: Enter the class name of the desired class to delete.

GOSD0081I Enter the name of a class :

Explanation: REGIMPL is prompting the user to enter the class name of the class to add to the naming service

User Response: Enter the class name of the desired class to add.

GOSD0083I Enter server class name (default: ***string*****) :**

Explanation: REGIMPL is prompting the user to enter the server class name for the server that is currently being added to the implementation repository through an add operation

User Response: Enter the desired server class name. The default is SOMDServer

GOSD0088E Error: the '*string*' option was entered more than once.

Explanation: The specified option can only be specified once.

User Response: Correct the option specification.

GOSD0089E Error: the '*string*' option should not have an argument.

Explanation: The specified option should not have an argument.

User Response: See chapter on registering classes and servers in Planning: SOMobjects for MVS for a list of valid options.

GOSD0092E Error: The alias name '*string*' was not found.

Explanation: The specified alias name was not found in the implementation repository.

User Response: Use the REGIMPL utility to correct the server implementation. For information about how to use REGIMPL to register a server, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0093E Error: The alias name '*string*' is already defined; choose another.

Explanation: The specified alias name is already defined in the implementation repository.

User Response: Use the REGIMPL utility to select another alias. For information about how to use REGIMPL, see Planning: SOMobjects for MVS.

GOSD0094E Error: The SOMDDIR environment setting is not defined.

Explanation: The SOMDDIR environment variable must specify the location of the implementation repository files.

User Response: Update the SOMDDIR environment variable so it specifies the correct location of the implementation repository files.

GOSD0096E Error occurred when initializing the Implementation Repository.

Explanation: REGIMPL failed while instantiating an implementation repository object.

User Response: Ensure that there is sufficient storage for the program to run. Verify that user has read access to the implementation repository. If the problem persists, contact the IBM Support Center.

GOSD0097E Error: An invalid operation was specified.

Explanation: An invalid operation was specified.

User Response: Correct the operation specification.

GOSD0099E Error: Unable to get the global Environment structure.

Explanation: The global environment could not be accessed.

User Response: Ensure that the SOMENV dataset is available. The dataset must be allocated as the SOMENV file is either a DDNAME of SOMENV in a JCL DD statement, or a file of SOMENV in a TSO ALLOCATE command.

GOSD0103E An error occurred when adding the class.

Explanation: DSOM found an error while adding a class to the implementation repository.

User Response: Correct the failing condition

GOSD0104E An initialization error occurred; User Exception: ***string*****.**

Explanation: Unable to initialize the DSOM environment.

User Response: Fix the error indicated in the message.

GOSD0108E An error occurred when deleting the server.

Explanation: An error was detected when deleting an implementation repository. The implementation repository to be deleted was not found.

User Response: Ensure that the specified implementation repository pointed by the SOMDDIR environment variable.

GOSD0111I Server '*string*' was successfully added.

Explanation: An implementation was successfully added.

User Response: None required.

GOSD0112I Server '*string*' was successfully deleted.

Explanation: An implementation was successfully deleted.

User Response: None required.

GOSD0113I Server '*string*' was successfully updated.

Explanation: An implementation was successfully updated.

User Response: None required.

GOSD0117I Operation was aborted at user's request.

Explanation: The user requested that the system end the operation.

User Response: None required.

GOSD0118I New alias :

Explanation: REGIMPL is prompting the user to enter the new alias name for the server that was requested to be updated in the implementation repository

User Response: Enter the new alias name for the server. If the user does not want to change the alias name, press the ENTER key.

GOSD0120I New server class :

Explanation: REGIMPL is prompting the user to enter the new server class for the server that was requested to be updated in the implementation repository

User Response: Enter the new server class for the server. If the user does not want to change the server class, press the ENTER key.

GOSD0125I The above server is about to be added. Add? [y/n]

Explanation: REGIMPL is prompting the user to ensure that the information displayed above this message (GOSD0537I) is to be added to the implementation repository.

User Response: If correct and the user wants to proceed, enter `'y'`. Otherwise enter `'n'`

GOSD0126I The above server is about to be updated. Update? [y/n]

Explanation: REGIMPL is prompting the user to ensure that the information displayed above this message (GOSD0537I) is to be updated in the implementation repository.

User Response: If correct and the user wants to proceed, enter `'y'`. Otherwise enter `'n'`

GOSD0127W Warning: No servers are defined in the Implementation Repository.

Explanation: No implementations have been defined in the implementation database.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0130I

```
[ SERVER OPERATIONS ]
 1.  Add       2.  Delete    3.  Change
 4.  Show one  5.  Show all  6.  List aliases

[ CLASS OPERATIONS ]
 7.  Add       8.  Delete    9.  Delete from all  10. List classes
 11. Add to all


12. SAVE and EXIT

Enter an operation:
```

Explanation: This message is the main menu for the DSOM Implementation Registration Utility (REGIMPL). It has 12 actions that a user can take on the implementation repository:

1. Add — Add a server
2. Delete — Delete a server
3. Update or change — Update a server
4. Show one — Show the information for one user-specified server
5. Show all — Show information for all servers in the implementation repository
6. List aliases — Show all the alias names of all the servers in the implementation repository
7. Add — Add a class associations to a server
8. Delete — Delete a class association to a server
9. Delete from all — Delete a class association from all servers in the implementation repository
10. List classes — List classes associated with all servers in the implementation repository
11. Add to all — Add a class association to all servers in the implementation repository
12. Save and exit — Exit the REGIMPL utility

User Response: Enter the number of the desired action to be performed by the REGIMPL utility.

GOSD0132I DSOM IMPLEMENTATION REGISTRATION UTILITY
(C) Copyright IBM Corp. 1992,1997. All rights reserved.

Explanation: This is a copyright statement

User Response: None required

GOSD0133E Error: Unknown option -%c was specified.

Explanation: User entered the REGIMPL command with an incorrect option.

User Response: Specify a valid option on the REGIMPL command. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for a list of valid options.

GOSD0134E Error: An action option must be specified.

Explanation: An action option must be specified.

User Response: Specify a valid option on the REGIMPL command. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for a list of valid options.

GOSD0135E Error: One implementation (-i) must be specified for the *string* action option.

Explanation: One implementation alias name must be specified with the option shown in the message text.

User Response: Enter the REGIMPL command again, specifying one implementation. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for syntax information.

GOSD0136E Error: The `'`*string*`'` option may only take `'on'` or `'off'` argument.

Explanation: User entered the REGIMPL command with an incorrect argument. The specified option accepts only `'on'` or `'off'` as a valid argument.

User Response: Enter the REGIMPL command again, specifying `'on'` or `'off'` as an argument. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for syntax information.

GOSD0139I No classes are defined.

Explanation: While printing implementation aliases and their classes, no classes were found.

User Response: None required.

GOSD0140E Error: At least one implementation alias (-i) must be specified for the *string* action option.

Explanation: One or two implementation alias names must be specified on the noted option shown in the message text.

User Response: Enter the REGIMPL command again, specifying one or two implementation aliases. See the section registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for syntax information.

GOSD0142E Error: Only an implementation alias (-i) may be specified for the *string* action option.

Explanation: Only implementation alias names may be specified on the noted action option.

User Response: Enter the REGIMPL command again, specifying implementation alias names with the option shown in the message text. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for syntax information.

GOSD0143E Error: At least one class (-c) must be specified for the *string* action option.

Explanation: At least one class name must be specified on the noted action option.

User Response: Enter the REGIMPL command again, specifying at least one class name with the option shown in the message text. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for syntax information. Specify at least one class name with the noted action option.

GOSD0144E Error: At least one implementation alias (-i) must be specified for the *string* action option.

Explanation: At least one implementation alias name must be specified on the option shown in the message text.

User Response: Specify at least one implementation alias name. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for syntax information.

GOSD0145E Error: An invalid item option was specified on the *string* action option.

Explanation: An invalid item option was specified on the noted action option.

User Response: Enter the REGIMPL command again, specifying a valid item option. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for a list of valid item options.

GOSD0148E Error: The '*string*' option requires an argument.

Explanation: The specified option requires an argument.

User Response: Enter the REGIMPL command again, specifying the required argument. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for syntax information.

GOSD0149E Error: More than one action option was specified.

Explanation: User entered a REGIMPL command with more than one action option. Only one action option may be specified at a time.

User Response: Enter the REGIMPL command again, specifying only one action option. See the section on registering classes and servers in the implementation repository in Planning: SOMobjects for MVS for a list of valid action options.

GOSD0150I SOMDD - Ready

Explanation: The DSOM daemon is ready to process requests from client and server programs.

User Response: None required.

GOSD0151I SOMDD - Ended

Explanation: The DSOM daemon has been stopped.

User Response: If desired, ask the system programmer to enter the DSOM START command to start the daemon again.

GOSD0162E SOMDD - ERROR: Invalid parameter; a valid parameter is -q.

Explanation: An invalid parameter has been based to the DSOM daemon.

User Response: Correct the parameter and restart the server.

GOSD0164E Error: Reference and backup files must have different names.

Explanation: When using the REGIMPL utility, user specified the same reference data file name and backup reference data file name for a server. These names must be different.

User Response: Specify different names for the reference data file name and its backup. See Planning: SOMobjects for MVS for information about how to enter the file names using the REGIMPL utility.

GOSD0165E Error: Cannot have a backup file without a reference file.

Explanation: When using the REGIMPL utility, user specified a backup reference data file name without first specifying a primary reference data file name. The primary file name must be specified first.

User Response: Specify different names for the reference data file name and its backup. See Planning: SOMobjects for MVS for information about how to enter the file names using the REGIMPL utility.

GOSD0167W WARNING: The directory for the reference file does not exist.

Explanation: The directory for the reference file does not exist.

User Response: If the result is as expected, no action is required. Otherwise, correct implementation database to specify the correct directory, or create the specified directory on user's system.

GOSD0168W WARNING: The directory for the backup file does not exist.

Explanation: The directory for the reference backup file does not exist.

User Response: If the result is as expected, no action is required. Otherwise, correct implementation database to specify the correct directory, or create the specified directory on user's system.

GOSD0169E Error: No more than 16 implementation aliases (-i) can be given at one time.

Explanation: When using the REGIMPL utility to register servers, user's installation registered more than 16 implementation aliases.

User Response: Limit the number of implementation aliases to 16.

GOSD0170E Error: No more than 16 class names (-c) can be given at one time.

Explanation: There is a limit of 16 class names.

User Response: Limit the number of class names to 16.

GOSD0200I SOMDSVR - Ended

Explanation: SOMDSVR ended.

User Response: None required.

GOSD0201I SOMDSVR - Implementation ID *string* **is not registered.**

Explanation: The specified implementation is not defined.

User Response: Ensure that user entered the correct implementation identifier.

GOSD0202E SOMDSVR - SOMOA_execute_request_loop failure, exception = *string***.**

Explanation: A failure occurred during the execution of the object adapter request loop.

User Response: Contact the IBM Support Center.

GOSD0203E SOMDSVR - SOMOA_impl_is_ready failure, exception = *string***.**

Explanation: The SOMOA is not ready. Possible reasons are that the implementation repository is not found, no server class exists, or the server is disabled.

User Response: Check for the indicated reasons.

GOSD0204I SOMDSVR - Ready

Explanation: The DSOM server program SOMDSVR is ready to process client requests.

User Response: None required.

GOSD0208E SOMDSVR - An invalid argument was specified.

Explanation: An invalid argument was passed to SOMDSVR.

User Response: Check and correct the parameters passed to SOMDSVR.

GOSD0209E SOMDSVR - An invalid number of arguments was specified.

Explanation: An incorrect number of arguments were specified on the SOMDSRV invocation.

User Response: Correct the number of arguments passed to SOMDSRV. Specify one argument (implementation ID) or two arguments (-a followed by the alias name).

GOSD0212I SOMDSVR - Usage: somdsvr <implid> | -a <alias>

Explanation: This messages is issued whenever SOMDSRV was invoked with an invalid argument list.

User Response: See previously issued message for possible actions.

GOSD0213I SOMDSVR - Server alias *string* **is not registered.**

Explanation: The server alias supplied on the SOMDSRV invocation was not found in the Implementation Repository.

User Response: Use the regimpl utility to list the contents of the Implementation Repository. The server alias may have been misspelled on the invocation of SOMDSVR or it may not be in the Implementation Repository. Correct spelling or add the server to the Implementation Repository.

GOSD0220E An invalid error code was encountered.

Explanation: An error code less than the DSOM base error code was passed to the DSOM error routine

User Response: Contact the IBM Support Center.

GOSD0221E DSOM *string* **error:** *dstring* **[***string***] at** *string***:***dstring*

Explanation: This message shows debugging information issued by DSOM. It displays the exception name, error code, error name, and the C file and line number where the error was found.

User Response: See the description of the error code that corresponds to this message, which is provided in the message text.

GOSD0233W Unable to open message log file (*string***), output will go to stdout.**

Explanation: Message log file specified on SOMDMESSAGELOG environment variable could not be opened.

User Response: Check that the data set is allocated and has the correct RACF authorization

GOSD0234W Unable to close message log file (*string***).**

Explanation: DSOM cannot close the indicated data set

User Response: Check that the data set is allocated and has the correct RACF authorization.

GOSD0247I SOMDServer dispatching method *string***.**

Explanation: Show the dispatch method on a SOM object in DEBUG mode.

User Response: None required.

GOSD0251I USAGE: dsom { stop|list|start|restart|enable|disable } { impl_alias [impl_alias...] [*] | * }

Explanation: This message displays the DSOM command syntax.

User Response: None required. See Planning: SOMobjects for MVS for detailed syntax information.

GOSD0252I The server process was stopped.

Explanation: The requested server process has stopped.

User Response: None required.

GOSD0253I The server process is not running.

Explanation: The DSOM implementation server was not found. A command was issued to a server process that was not currently active.

User Response: Enter the REGIMPL LIST command to display a list of valid servers. Enter the command again, specifying a valid server name.

GOSD0254I Cannot stop server; user is not the process owner.

Explanation: A DSOM stop was issued for a server process and the server process was not started by the user issuing the stop or the user does not have proper authority.

User Response: If user still need to stop the specified server, obtain the proper authority from user's DSOM administrator.

GOSD0257I The client has timed out.

Explanation: Communications between the Client and Daemon or Server has timed out.

User Response: Do the following:

1. Ensure that the DSOM daemon address space is active.
2. Ensure that the HOSTNAME environment variable specifies the correct hostname.
3. Ensure that the requested server is active.
4. If necessary, increase the value specified on the SOMDTIMEOUT environment variable.

GOSD0258E A send error occurred.

Explanation: An error was received from a TCP/IP function

User Response: Check DSOM trace for error information received from TCP/IP

GOSD0259I Server activation is pending.

Explanation: The server is in the process of activating. The request has been queued and will be processed when the server becomes active.

User Response: None.

GOSD0260E Unable to initialize; SOMD_Init failed.

Explanation: Unable to initialize the environment. This message is mapped onto DSOM error code SOMDERROR_NoSOMDInit.

User Response: Use the IRDUMP utility to check that all interface repositories pointed to by SOMIR environment variable are valid. If the interface repositories are valid, contact the IBM Support Center.

GOSD0262E The request was unsuccessful; an unknown error occurred.

Explanation: The DSOM ENABLE command could not perform the requested operation.

User Response: Contact the IBM support center.

GOSD0263I The server process is currently running.

Explanation: The server process is currently running. This message is issued in response to the DSOM LIST utility command.

User Response: None required.

GOSD0264I The server process is currently disabled.

Explanation: The server process is currently disabled.

User Response: None required.

GOSD0265I The server process was started.

Explanation: The server process was started. This message is issued in response to the DSOM START utility command.

User Response: None required.

GOSD0266I The server process cannot be started.

Explanation: The DSOM daemon cannot start a new process to execute the server program. Server programs are specified in the Implementation Repository.

User Response: Ensure that the server program resides in the Implementation Repository pointed by the SOMDDIR environment variable.

GOSD0267I The server process was restarted.

Explanation: The server process was restarted. This is issued as a result of the DSOM RESTART utility command.

User Response: None required.

GOSD0268I The server was successfully disabled.

Explanation: User entered a DSOM DISABLE utility command. DSOM disabled the specified server.

User Response: None required.

GOSD0269I The server was successfully enabled.

Explanation: User entered a DSOM ENABLE command to enable a server. The specified server is enabled.

User Response: None required.

GOSD0270E *string@string*: **Server alias was not found in the Implementation Repository.**

Explanation: User entered a DSOM server utility command. DSOM could not process the command because the server alias was not found in the implementation repository.

User Response: Enter the command again, specifying a valid server alias name.

GOSD0271I The server process is marked for deletion; try again.

Explanation: User entered a DSOM server utility command. A previous request to shut down the server is pending.

User Response: Enter the command again. If the problem persists, contact the IBM Support Center.

GOSD0287W Warning: SOMDDIR not set. Using default value.

Explanation: SOMDDIR environment variable is not specified in the configuration file. System will use the default value for SOMDDIR which is specified in SOMBASE.

User Response: None required.

GOSD0329E The fopen failed for file *string*.

Explanation: Error occurred while opening the class data file or the class index file when using the MIGIMPL utility.

User Response: Check SOMDDIR environment variable is set to point to the Implementation Repository. Repository.

GOSD0343I Call *string* **returned return code of** *dstring*

Explanation: A call to the function (*string*) returned with the return code *dstring*.

User Response: None required.

GOSD0358I HOSTNAME = (null).

Explanation: Display hostname information.

User Response: None required.

GOSD0396I

```
Implementation Alias
====================
```

Explanation: Display an implementation alias in the implementation repository.

User Response: None required.

GOSD0397I Enter ImplDef class name (default: ImplementationDef):

Explanation: REGIMPL is prompting the user to enter the impldef class name to be used by the server that is being added to the implementation repository

User Response: Enter the class name. To take the default setting, press enter.

GOSD0399I New ImplDef class name:

Explanation: REGIMPL is prompting the user to enter a new impldef class name to be used by the server that is being updated in the implementation repository

User Response: Enter the new class name. If nothing is entered, the ImplDef class name will not be updated.

GOSD0401E Error: Cannot create ImplRepository object.

Explanation: REGIMPL failed while instantiating an implementation repository object.

User Response: Ensure that there is sufficient storage for the program to run. Verify that user has read access to the implementation repository. If the problem persists, contact the IBM Support Center.

GOSD0402W Warning: Server was registered, but Naming Service could not be updated. Naming Service update will be attempted upon next server update.

Explanation: Entry is added to the Implementation Repository without updating the Naming Service. Information in both the Implementation Repository and the Naming Service must kept consistent.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command.

GOSD0403E Error occurred when updating class entries in Naming Service.

Explanation: Error occurred when updating the implementation repository because the Naming Service is not active. Entries with information in both the Implementation Repository the Naming Service must be kept consistent. Such entries cannot be updated or deleted if the Naming Service is not active.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command.

GOSD0405E Error occurred when deleting class. Error code was *dstring*.

Explanation: Error occurred when deleting class entries from Naming Service because the Naming Service is not active. Entries with information in both the Implementation Repository and the Naming Service must be kept consistent. Such entries cannot be updated or deleted if the Naming Service is not active.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command.

GOSD0406E Error occurred when listing class.

Explanation: Error occurred when listing class. This can occur if the Naming Service is not active.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command.

GOSD0417E Error: Cannot create an instance of the *string* registrar class.

Explanation: Fail to create Callstream registar object from the protocol information specified in the configuration file.

User Response: Make sure that correct communication protocol information is specified in the configuration file.

GOSD0418E Error: Cannot display protocol information for the '*string*' protocol.

Explanation: Unable to display protocol information for the protocol specified.

User Response: Make sure that correct communication protocol information is specified in the configuration file.

GOSD0419E Error: Server has an invalid object reference. Reregister server.

Explanation: When printing protocol information, REGIMPL encountered an invalid object reference.

User Response: Remove the implementation definition entry for the server from the Interface Repository and reregister the server. This can be done by using the REGIMPL utility.

GOSD0420E Error: Instance of subclassed ImplementationDef cannot be created.

Explanation: Unable to create an instance of subclassed ImplementationDef.

User Response: Ensure that the subclassed ImplementationDef class has an entry in the Interface Repository and that the DLL name given by the dllname modifier can be loaded. (Use the IRDUMP facility to determine whether a particular class appears in the IR.)

GOSD0421E Error occurred when trying to show ImplementationDefs.

Explanation: Error occurred when trying to show all ImplementationDefs objects in the implementation repository. This can occur when the Naming Service is not active or implementation repository is not found.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command. Also ensure that the implementation repository is pointed by the SOMDDIR environment variable or the value specified in SOMBASE if SOMDDIR is not set.

GOSD0422E Error occurred when trying to show server aliases.

Explanation: Error occurred when trying to get all the all the impl_aliases from the Implementation Repository. This can occur when the Naming Service is not active or implementation repository is not found.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command. Also ensure that the implementation repository is pointed by the SOMDDIR environment variable or the value specified in SOMBASE if SOMDDIR is not set.

GOSD0423E Error: Invalid protocol information for the '*string*' protocol.

Explanation: Invalid protocol information is specified as SOMDPROTOCOLS environment variable.

User Response: Make sure that correct communication protocol information is specified in the configuration file. The value specified should be SOMD_TCPIP.

GOSD0425E Error: Cannot create an instance of the '*string*' class

Explanation: Unable to create an instance of the specified class.

User Response: Ensure that there is sufficient storage for the program to run. If the problem persists, contact the IBM Support Center.

GOSD0426E Error: Cannot create the '*string*' class object

Explanation: DSOM run time is failing to load the specified class, which can occur if the class name specified is not associated with any server that is registered.

User Response: Ensure that the class name is associated with at least one server.

GOSD0427E Error: Cannot create an ImplementationDef object for '*string*'.

Explanation: Unable to get the impldef object corresponding to the alias specified. This can occur when the Naming Service is not active or implementation repository is not found.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command. Also ensure that the implementation repository is pointed by the SOMDDIR environment variable or the value specified in SOMBASE if SOMDDIR is not set.

GOSD0428E Error: The Naming Server is not active.

Explanation: The Naming Server is not active.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command.

GOSD0429E Error: The Name was not found.

Explanation: The Naming Service cannot resolve the name, which can happen if the name was never bound to the naming context or if the name was unbound from the context.

User Response: Check if the server or class associated with the name has an entry in the implementation repository. If it does, remove the entry from the implementation repository and reregister the server or class. This can be done using the REGIMPL utility.

GOSD0430W Warning: The Name is already bound !

Explanation: The bind method is attempting to associate an object to a name that was previously bound in the same context.

User Response: No action required.

GOSD0431E Error: Cannot access the Implementation Repository.

Explanation: Unable to access the Implementation Repository. This might indicate that the Implementation Repository files cannot be found, cannot be accessed, might be corrupted.

User Response: Follow these steps:

1. Verify that the SOMDDIR environment variable is set properly to the high level qualifier of the files.
2. If the SOMDDIR environment variable is not set, verify the default value for SOMDDIR which is specified in SOMBASE.
3. Ensure that all the implementation repository files have read and write permissions granted to the DSOM user.

GOSD0432E Error: Cannot find the specified server alias.

Explanation: The specified server alias in the Implementation Repository cannot be found.

User Response: Use the regimpl command to examine the contents of the implementation repository. Reissue the command with the correct alias.

GOSD0433E Error occurred when adding class entries to the Naming Service.

Explanation: Error occurred when adding class entries to the Naming Service. This can occur when the Naming Service is not active.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command.

GOSD0434I The class '*string*' is now associated with all servers.

Explanation: The specified class is now associated successfully with all servers.

User Response: None required.

GOSD0435E Error occurred when removing class '*string*' from server *string*.

Explanation: Error occurred when removing the specified class from the specified server in the Naming Service because the Naming Service is not active. Entries with information in both the Implementation Repository and the Naming Service must be kept consistent. Such entries cannot be updated or deleted if the Naming Service is not active.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command. Also ensure that the daemon and naming server can communicate each other.

GOSD0436I The class '*string*' has been removed from server *string*.

Explanation: The specified class has been removed successfully from the specified server.

User Response: None required.

GOSD0437E Error occurred when removing class '*string*' from the Naming Service.

Explanation: Error occurred when removing the specified class from the Naming Service because the Naming Service is not active. Entries with information in both the Implementation Repository and the Naming Service must be kept consistent. Such entries cannot be updated or deleted if the Naming Service is not active.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command. Also ensure that the daemon and naming server can communicate each other.

GOSD0438I Class '*string*' has been removed from the Naming Service.

Explanation: The specified class has been removed successfully from the Naming Service.

User Response: None required.

GOSD0440I REGIMPL processing has completed.

Explanation: The REGIMPL utility has completed all processing

User Response: None required.

GOSD0442E Error: An invocation of lookup_modifier failed.

Explanation: Lookup_modifier failed when trying to set the values of the additional attributes in a subclassed ImplementationDef. This can occur if the object does not exist or does not possess the modifier.

User Response: Use the regimpl command to examine the contents of the implementation repository. Reissue the command with the correct attribute name.

GOSD0443E Error: Method '*string***' is not supported.**

Explanation: This may indicate that the interface describing the method of the subclassed ImplemetationDef object cannot be found in the Interface Repository.

User Response: Verify that SOMIR is set correctly, and that the IR contains all interfaces used by the subclassed ImplemetationDef class.

GOSD0445E Error: Protocol *string* **is not supported.**

Explanation: Invalid protocol information is being specified in the configuration file.

User Response: See "Configuration and Startup" for a description of SOMDPROTOCOLS and stanza definitions for each protocol.

GOSD0446E Error occurred when adding class; error code was *dstring***.**

Explanation: Error occurred when adding a new class to an ImplementationDef.

User Response: Refer to the DSOM Error codes and correct the failing conditions based on the error code.

GOSD0447E Error occurred when adding class *string* **to server** *string***.**

Explanation: Error occurred when adding a new class to the specified server. This can occur if the Naming Service was not active.

User Response: Verify that som@cfg has been run by checking for HOSTKIND and SOMNMOBJREF entries in the [somnm] stanza of the configuration file. Ensure that the naming service is active. If it is not, start the naming server and reissue the REGIMPL command.

GOSD0455I *string*

Explanation: Displays a part of the protocol information.

User Response: None required.

GOSD0456I *string***:** *string*

Explanation: Displays a part of the protocol information.

User Response: None required.

GOSD0458I Select protocol '*string***'? [y/n] (default: yes):**

Explanation: REGIMPL is prompting the user to state whether or not the indicated protocol will be supported by the server being added or updated.

User Response: Enter `'y'` if this protocol should be supported by this server. Otherwise, enter `'n'`. To take the default setting, press enter.

GOSD0460I Make server secure [y/n] (default: no):

Explanation: REGIMPL is prompting the user to state whether or not the server being added should be secure.

User Response: Enter `'y'` if the server should be secure. Otherwise, enter `'n'`. To take the default, press enter.

GOSD0463I Make server secure? [y/n]:

Explanation: REGIMPL is prompting the user to state whether or not the server being updated should be secure.

User Response: Enter 'y' if the server should be secure. Otherwise, enter 'n'. If nothing is entered, the setting will not be updated.

GOSD0465I Update protocol information? [y/n] (default: no):

Explanation: REGIMPL is prompting the user to indicate whether the protocol information for this server should be updated in the implementation repository

User Response: If the protocol information should be updated, enter 'y'. Otherwise, enter 'n'. To take the default, press the enter key.

GOSD0466I The above class is about to be added. Add? [y/n]:

Explanation: REGIMPL is prompting the user to ensure that the information displayed above this message (GOSD0538I) is to be added to the implementation repository.

User Response: If correct and the user wants to proceed, enter 'y'. Otherwise enter 'n'

GOSD0467I Enter additional property <name value> pairs:

Explanation: REGIMPL is prompting the user to enter name and properties for the class to be stored in the naming service.

User Response: Enter the <name value> pair for the property. For example --> owner joe. If no more properties are desired, press the enter key.

GOSD0468I Enter value for '*string*':

Explanation: REGIMPL is prompting the user to enter in values for the class that is a subclass or the ImplementationDef class

User Response: Enter in the indicated information for the subclassed ImplementationDef

GOSD0471I

```
DSOM 2.1 to DSOM 3.0 IMPLEMENTATION REPOSITORY MIGRATION TOOL
(C) Copyright IBM Corp. 1992,1997.  All rights reserved.
Command syntax to convert DSOM 2.1 Implementation Repository
 entries to DSOM 3.0 format:

       migimpl3 [-l] [-U] [-i <str> [-i ...]]

The default action is to convert all entries in the 2.1 Repository
whose impl_hostname attribute matches the current HOSTNAME setting.

Servers are registered for the SOMD_TCPIP protocol, and for the protocol
corresponding to the current SOMSOCKETS setting, if any.

Existing 3.0 Repository entries are not updated unless the -U option is used.
2.1 entries for 'localhost' are not migrated unless the -l option is used.

    -l          = Migrates entries whose impl_hostname is 'localhost'
                  or matches current HOSTNAME setting.


    -U          = Only updates existing entries in 3.0 Repository.
                  (Does not add new entries.)
                  Updates are performed only for entries whose impl_hostname
                  matches current HOSTNAME setting or is 'localhost'.

    -i <str>    = Implementation alias name
```

Explanation: MIGIMPL3 help text.

User Response: None required.

GOSD0486E fseek failed for *string*.

Explanation: MIGIMPL failed when trying to fseek the implementation repository related files to get class names along with all the implementation aliases.

User Response: Follow these steps:

1. Verify that the SOMDDIR environment variable is set properly to the high level qualifier of the files.
2. If the SOMDDIR environment variable is not set, verify the default value for SOMDDIR which is specified in SOMBASE.
3. Ensure that all the implementation repository files have read and write permissions granted to the DSOM user.

GOSD0487E fread failed for *string*.

Explanation: MIGIMPL failed when trying to fread the implementation repository related files to get class names along with all the implementation aliases.

User Response: Follow these steps:

1. Verify that the SOMDDIR environment variable is set properly to the high level qualifier of the files.
2. If the SOMDDIR environment variable is not set, verify the default value for SOMDDIR which is specified in SOMBASE.
3. Ensure that all the implementation repository files have read and write permissions granted to the DSOM user.

GOSD0488E Memory allocation failed.

Explanation: DSOM run time is failing to allocate the necessary memory.

User Response: Ensure that there is sufficient storage for the program to run. If the problem persists, bring down the DSOM application and free system resources.

GOSD0489E Update failed: server has a different implementation ID *string* in 3.0 Repository.

Explanation: Migration failed to update an impldef because duplicate Alias Name exists in the 3.0 repository and the 3.0 implid is not the same as the 2.0 implid.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0490E Migration failed: Duplicate alias name *string* exists in 3.0 Repository.

Explanation: Migration failed to add an impldef because duplicate Alias Name exists in the 3.0 repository.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0491E Migration failed: Duplicate implid *string* exists in 3.0 Repository.

Explanation: Migration failed to add an impldef because the 3.0 implid is the same as the 2.0 implid.

User Response: If the result is as expected, no action is required. Otherwise, use the REGIMPL utility to correct the implementation data base. For information about how to use REGIMPL, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

GOSD0501E Error: Protocol list exhausted. Check setting of SOMDPROTOCOLS.

Explanation: Invalid protocol information is being specified in the configuration file.

User Response: See "Configuration and Startup" for a description of SOMDPROTOCOLS and stanza definitions for each protocol.

GOSD0515E Marshaling Error: Trying to marshal a union with an invalid discriminator value: ***dstring*****.**

Explanation: Marshaling error occurred when trying to marshal a union. The union discriminant value does not match any of the defined cases and no default case is defined.

User Response: See Union Type in *OS/390 V2R4.0 SOMobjects Programmer's Guide* for additional information on the union IDL type.

GOSD0516E TypeCode conversion failed while marshaling an Any.

Explanation: TypeCode conversion failed during Marshaling. The type code is being encountered while processing the request.

User Response: Ensure a valid type code is being used.

GOSD0517W Warning freeing Environment: Unexpected exception parameters.

Explanation: Unexpected exception parameters are being encountered when freeing the non-contiguous parts of Environment structure.

User Response: None required. See interp_environment for the description of an Environment.

GOSD0518E Marshaling Error: Trying to free a union with an invalid discriminator value: ***dstring*****.**

Explanation: Marshaling failed when trying to free a union. The union discriminant value does not match any of the defined cases and no default case is defined.

User Response: See Union Type in *OS/390 V2R4.0 SOMobjects Programmer's Guide* for additional information on the union IDL type.

GOSD0519E TypeCode conversion failed while freeing an Any.

Explanation: TypeCode conversion failed during Marshaling. An invalid type code is being encountered while processing the request.

User Response: Ensure that a valid type code is being used.

GOSD0520E An invalid TypeCode was passed to SOMD_FreeType

Explanation: TypeCode conversion failed during Marshaling. An invalid type code is being encountered while processing the request.

User Response: Ensure that a valid type code is being used.

GOSD0521E Error occurred while creating a marshal plan for method ***string*****.**

Explanation: Error occurred while creating a marshal plan for the specified method. An invalid type code is being encountered while processing the request.

User Response: Ensure that a valid type code is being used.

GOSD0522E SOMDDataMarshaler failed during TypeCode conversion.

Explanation: TypeCode conversion failed during Marshaling. An invalid type code is being encountered while processing the request.

User Response: Ensure that a valid type code is being used.

GOSD0523E SOMDDataMarshaler failed during marshaling.

Explanation: The data passed to the DSOM marshaler is not valid or did not match the expected data type.

User Response: Follow these steps:

1. Ensure that the data passed to the DSOM marshaler (for example, the parameters passed to a remote invocation) are correctly initialized.
2. Ensure that all data structures are constructed according to the data type definitions in IDL for the method being invoked.

GOSD0524E SOMDDataMarshaler failed during demarshaling.

Explanation: The data passed to the DSOM marshaler is not valid or did not match the expected data type.

User Response: Follow these steps:

1. Ensure that the data passed to the DSOM marshaler (for example, the parameters passed to a remote invocation) are correctly initialized.
2. Ensure that all data structures are constructed according to the data type definitions in IDL for the method being invoked.

GOSD0527E Interpretive marshaler encountered a type, *dstring*, that is not yet implemented!

Explanation: Interpreted marshaler encountered a type that is not yet implemented. An internal DSOM error has occurred.

User Response: Retry the scenario. If the problem persists, contact the IBM Support Center.

GOSD0528E Invalid discriminator size in union: *dstring*

Explanation: Interpreted marshaler encountered an invalid discriminator size in union. An internal DSOM error has occurred.

User Response: Retry the scenario. If the problem persists, contact the IBM Support Center.

GOSD0529E The DSOM daemon was unable to start process *string***.**

Explanation: The DSOM daemon could not start a new process to execute the server program. Server programs are specified in the Implementation Repository.

User Response: Ensure that the server program resides in the Implementation Repository pointed by the SOMDDIR environment variable.

GOSD0530E Class *string* **could not be loaded.**

Explanation: One of the following applies: DSOM run time is failing to load the specified class, which can occur if the class name specified is not associated with any server that is registered. The class libraries (DLLs) used to build the proxy class are statically linked to the program, but the DLL's SOMInitMoulde function is not properly initializing the class object, or has no SOMInitModule.

User Response: Follow these steps:

1. Ensure that the class name is associated with at least one of the server implementation.
2. Ensure that the class has an entry in the Interface Repository.
3. Verify that the DLL contains the SOMInitModule initialization function.
4. Ensure that the IDL for the class contains the dllname modifier, that this IDL has been compiled into the Interface Repository, and that the DLL name given by the dllname modifier can be loaded. (Use the irdump utility to determine whether a particular class appears in the IR.)

GOSD0532E Invalid parameter *string* was passed to a function or method.

Explanation: An invalid parameter value is being passed to a function or method.

User Response: See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for a description of the parameters to the specified function or method.

GOSD0533E The configuration setting *string* has an invalid setting.

Explanation: The indicated configuration variable setting is not valid.

User Response: See *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* for a description of the environment variable and " set it to a valid value.

GOSD0534E The exception *string* cannot be raised (it is not specified in the IDL).

Explanation: The application is raising an exception that is not specified in the method definition.

User Response: Modify the method IDL to contain a raises expression for the exception.

GOSD0535I Error adding reserved property identifiers. Class, alias, serverId, and serverRef are four reserved property identifiers.

Explanation: None.

User Response: Contact IBM support.

GOSD0536I Error removing default class _NULL.

Explanation: None.

User Response: Contact IBM support.

GOSD0537I

```
======================================================================

Implementation ID.........:
Implementation alias......:
ImplDef class name........:
Server secure.............:
Server class..............:
Protocol information......:
   Protocol:           ;Hostname:                    ;Port:      ;

======================================================================
```

Explanation: This message displays the information about a server that is being added, updated, or requested to be displayed. The display contains the following information:

Implementation id
: Implementation ID of the server

Implementation alias
: Implementation alias of the server

ImplDef class name
: Name of the class or subclass for this server

Server secure
: An indication whether the server should accept requests from authenticated clients only

Server class
: The name of the SOMDServer class or subclass that will manage the objects in the server.

Protocol Information
: Displays the protocol, Hostname, and port of each protocol supported by this server
: **Protocol:** *string*;
: **Hostname:** *string*;
: **Port:** *dstring*;

Additional Attributes
    Specific ImplementationDef subclass information.

User Response: None required.

GOSD0538I

```
==================================================================

Implementation ID.........:
Implementation alias......:
Class name................:
   Property / Value.......:

==================================================================
```

Explanation: This messages displays the information about a class association to a server and any property values associated with the class. This message displays the following information:

Implementation id
    Implementation id of the server
Implementation alias
    Implementation alias of the server
Class name
    name of the class to be associated with the server
Property / Value
    User defined properties that were specified when the class was associated to this server There may be one or more of these listed.

User Response: None required.

GOSD0539E The DSOM daemon was unable to start process *string* **RC** *dstring* **RS** *dstring* **.**

Explanation: The DSOM daemon could not start a process. The process must be defined in the Implementation
| Repository and in the WLM Service Definition. The decimal return code, RC, and the hexadecimal reason code, RS,
| were returned from the WLM IWMSRFSV macro.

User Response: Ensure that the server program resides in the Implementation Repository pointed by the SOMDDIR environment variable, have the WLM service administrator ensure that the implementation alias is defined as a WLM application environment, have the Security service administrator ensure that the alias is defined in the security facility, check that the related procedure is in the proclib and check if the procedure received a jcl error. The WLM decimal return code and hexadecimal reason code can assist in providing a more precise error identification. See the IWMSRFSV macro description in the OS/390 MVS Workload Management Services manual for the return code and reason code translation if the error was reported by a DSOM server.

GOSD0540I MIGIMPL3 processing has completed.

Explanation: The MIGIMPL3 utility has completed all processing

User Response: None required.

GOSD0541E Could not create enclave RC *dstring* **RS** *dstring***.**

| **Explanation:** The attempt to create an enclave failed. The decimal return code, RC, and the hexadecimal reason
| code, RS, were returned from the IWMECREA macro.

| **User Response:** See the IWMECREA macro description in the OS/390 MVS Workload Management Services
| manual for the decimal return code and hexadecimal reason code translation if the error was reported by a DSOM
| server.

GOSD0542E Could not join enclave RC *dstring* RS *dstring*.

| **Explanation:** The attempt to join an enclave failed. The decimal return code, RC, and the hexadecimal reason code,
| RS, were returned from the IWMEJOIN macro.

| **User Response:** See the IWMEJOIN macro description in the OS/390 MVS Workload Management Services manual
| for the decimal return code and hexadecimal reason code translation if the error was reported by a DSOM server.

GOSD0543E Could not leave enclave RC *dstring* RS *dstring*.

| **Explanation:** The attempt to leave an enclave failed. The decimal return code, RC, and the hexadecimal reason
| code, RS, were returned from the IWMELEAV macro.

| **User Response:** See the IWMELEAV macro description in the OS/390 MVS Workload Management Services
| manual for the decimal return code and hexadecimal reason code translation if the error was reported by a DSOM
| server.

GOSD0544E Could not delete enclave RC *dstring* RS *dstring*

| **Explanation:** The attempt to delete an enclave failed. The decimal return code, RC, and the hexadecimal reason
| code, RS, were returned from the IWMEDELE macro.

| **User Response:** See the IWMEDELE macro description in the OS/390 MVS Workload Management Services
| manual for the decimal return code and hexadecimal reason code translation if the error was reported by a DSOM
| server.

GOSD0545E Error: Invalid alias name -- Contains all blanks or is NULL.

Explanation: The alias name entered contains all blanks

User Response: Reissue the REGIMPL request and supply a valid alias name

GOSD0546E Error: Invalid alias name -- An invalid character was found. Alias name can contain the alphanumeric (a-z, A-Z,0-9), national (@,#, $), and the underscore characters.

Explanation: The alias name entered contains an invalid character. The valid characters accepted are found the error message.

User Response: Reissue the REGIMPL request and supply a valid alias name

GOSD0547E Error: Invalid alias name -- Embedded blanks are not allowed.

Explanation: The alias name entered contains embedded blanks

User Response: Reissue the REGIMPL request and supply a valid alias name

GOSD0548E Error: Invalid alias name -- Length cannot exceed 32 characters.

Explanation: The alias name entered is longer than 32 characters

User Response: Reissue the REGIMPL request and supply a valid alias name

GOSD0549E Error: Invalid alias name -- Cannot start with "SYS".

Explanation: The alias name entered cannot start with "SYS".

User Response: Reissue the REGIMPL request and supply a valid alias name

GOSD0550E Error: Invalid alias name.

Explanation: The alias name entered is invalid.

User Response: Reissue the REGIMPL request and supply a valid alias name

GOSD0551E Error: Invalid class name -- Contains all blanks or is NULL.

Explanation: The class name entered contains all blanks

User Response: Reissue the REGIMPL request and supply a valid class name

GOSD0552E Error: Invalid class name -- Embedded blanks are not allowed.

Explanation: The class name entered contains embedded blanks

User Response: Reissue the REGIMPL request and supply a valid class name

GOSD0553E Error: Invalid class name.

Explanation: The class name entered is invalid.

User Response: Reissue the REGIMPL request and supply a valid class name

GOSD0554I Enter migimpl3 -? for help

Explanation: Enter migimpl3 -? for help.

User Response: None required.

GOSD0555E Error: The alias name '*string*' was not migrated.

Explanation: The alias mentioned was not migrated

User Response: Enter migimpl -? for help and check that the alias name meets the correct name conventions

GOSD0556E Error: The DSOM daemon is being stopped. Process '*string*' was not started.

Explanation: The DSOM daemon did not start a new process to execute the server program because the daemon is being stopped.

User Response: Wait for the daemon to be stopped and restarted before restarting the program.

GOSD0557E Error: The DSOM daemon is being stopped. Process '*string*' was not connected.

Explanation: The server process did not connect to WLM because the daemon is being stopped.

User Response: Wait for the daemon to be stopped and restarted before restarting the program.

GOSD0558E Error: Socket error '*dstring*' encountered reading message header.

Explanation: An attempt to read a IIOP message header encountered an error.

User Response: Use the socket error code to determine the cause of the problem.

Core Services Messages

Following are the texts, explanations, and user responses for messages issued by the SOMobjects Core Services. These messages appear in the program listing.

GOSJ0001E Operation '*string1*' failed.

Explanation: An unexpected error occurred while processing the operation identified by the message. The executing process will terminate.

User Response: Report problem to IBM.

GOSJ0002E Resource '*string1*' invalid.

Explanation: The meta-state database key or database name identified by the message is in error. The executing process will terminate.

User Response: Report problem to IBM.

GOSJ0003E Resource '*string1*' not found.

Explanation: The meta-state database identified in the message could not be found. The executing process will terminate.

User Response: Report problem to IBM.

GOSJ0004E Resource '*string1*' already exists.

Explanation: The database identified by the message already exists. The executing process will terminate.

User Response: If this problem occurs while running SOM@CFG, delete the "SOMOSDB.DAT" database and restart SOM@CFG. Otherwise, report problem to IBM.

GOSJ0005E I/O error on '*string1*'.

Explanation: An I/O error occurred while accessing the indicated data set or file. The executing process will terminate.

User Response: Initiate standard problem determination procedure for I/O errors.

GOSJ0007E Operation '*string1*' ran out of memory allocating *string2* bytes.

Explanation: There was insufficient memory to perform the indicated operation. The executing process will terminate.

User Response: Increase region and heap sizes, then restart process.

GOSJ0008E System problem: '*string10*'.

Explanation: The indicated system problem occurred. The executing process will terminate.

User Response: Initiate standard problem determination. Restart process if problem found and corrected. If problem cannot be determined, contact IBM.

Kernel Messages

Following are the texts, explanations, and user responses for messages issued by the SOMobjects kernel.

SOMobjects returns one of the messages described in this section when it finds an error in the most recently completed call to a SOMobjects service. These messages appear in the program listing.

GOSK0001W Warning: NULL class argument passed to SOMClass::somDescendedFrom.

Explanation: A class was not passed to the somDescendedFrom method.

User Response: Update the argument to point to a SOM class.

GOSK0002E Error: The internal buffer used in somPrintf overflowed.

Explanation: The string passed as an argument into the somPrintf function is longer than 4095 characters.

User Response: Reduce the length of the string passed into the somPrintf function.

GOSK0003E Error: SOMClass::somFindMethodOk failed to find a method.

Explanation: The somFindMethod failed to find the method requested.

User Response: Check the program logic to ensure the correct method is being requested.

GOSK0005E Error: somDefaultMethod was called. (This probably means a defined method was invoked before it was added to the class.)

Explanation: A defined method was invoked before it was added to the specified class.

User Response: Check program logic to ensure the correct method is added to the specified class.

GOSK0006E Error: Current method not defined on the target object.

Explanation: The method invoked was not one of the methods supported by the target object.

User Response: Check program logic to ensure the intended object method is being used.

GOSK0007E Error: Attempt to load, create or use an incompatible class.

Explanation: The program attempted to use a class that had a different version number from the one requested.

User Response: Update the program logic to access the intended version of the class.

GOSK0009E Error: Memory exhausted.

Explanation: The program used more virtual storage than was available.

User Response: Increase the region size.

GOSK0010E Error: A target object failed basic validity checks during method resolution.

Explanation: The object did not pass a basic validity check. A prior message explains the reason for the failure.

User Response: Check program logic.

GOSK0011E Error: A user-defined test condition failed.

Explanation: The program failed a condition coded on the SOM_Test macro. See message GOSK0036E for more information.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for information about the SOM_Test macro.

GOSK0012W Warning: SOMClassMgr::somFindClass failed.

Explanation: The SOM class manager was unable to locate the specified class.

User Response: Check your program logic to ensure the class is accessible.

GOSK0014E Error: SOM runtime initialization failed to complete.

Explanation: The SOM kernel was not able to create the three base class objects.

User Response: Check for other error messages for information on the cause of the error.

GOSK0015E Error: SOMClassMgr::somUnloadClassFile failed.

Explanation: The SOM class manager was unable to unload the class file because it was not registered.

User Response: Check program logic to ensure the correct class file is being unloaded.

GOSK0016E Error: SOMClass::somOverrideSMethod called for a method that was not defined in a parent class.

Explanation: The static method could not be overridden because it was not defined in the object's parent.

User Response: Check your program logic to ensure the correct object and method were specified.

GOSK0018E Error: Subclass failed to override a method.

Explanation: The virtual method must be overridden in the subclass to provide an implementation.

User Response: Correct your program logic to override the method.

GOSK0019E Error: An argument failed a validity test.

Explanation: The argument passed on a SOM method or function was incorrect.

User Response: Correct the error as directed by a prior error message.

GOSK0021E Error: While creating a class object the parent class could not be found.

Explanation: The SOM Kernel could not locate the object's parent class during object creation.

User Response: Check your program logic to ensure that the parent class is set up correctly.

GOSK0023E Error: Attempted to index an out-of-range buffer entry.

Explanation: The number of static methods added to a class exceeded the maximum number of static methods supported by the class.

User Response: Check your program logic.

GOSK0024E Error: Attempted to delete a character from an empty buffer.

Explanation: Internal SOM runtime library error.

User Response: Contact your system programmer.

GOSK0025E Error: Internal logic error during buffer manipulation.

Explanation: Internal SOM runtime library error.

User Response: Contact your system programmer.

GOSK0028W Warning: \"*string***\":** *dstring***: SOM Error - code =** *dstring***-% 3d-***dstring***, severity = Fatal.**

Explanation: The SOM runtime library encountered a fatal error.

User Response: Correct the error as directed by other SOM error messages. See the SOM Kernel Error Codes for more information.

GOSK0029W Warning: Cannot obtain %ld bytes in *string***.**

Explanation: The program ran out of virtual storage.

User Response: Check your program logic. You can also try running the program with a larger region size.

GOSK0030W Warning: \"*string***\":** *dstring***: Object was NULL.**

Explanation: Method validity checking (established through the SOM_teston directive) detected that the specified object was NULL.

User Response: Correct your program to use a valid object.

GOSK0031W Warning: \"*string***\":** *dstring***: Object is not initialized or it has been freed.**

Explanation: Method validity checking (established through the SOM_teston directive) detected that the specified object has not been initialized or has been freed.

User Response: Correct your program logic.

GOSK0032W Warning: \"*string***\":** *dstring***: Class object was NULL.**

Explanation: The object failed to point to a valid class object.

User Response: Correct your program logic.

GOSK0033W Warning: \"*string***\":** *dstring***: Object was not of the expected class \"***string***\".**

Explanation: Method validity checking (established through the SOM_teston directive) detected that the object is not a descendant of the specified object. Method validity checking was established through the SOM_teston directive.

User Response: Correct your program logic.

GOSK0034I \"*string***\":** *dstring***: Passed -** *string***.**

Explanation: The program passed the condition specified on a SOM_Test, SOM_TestC, or SOM_Expect macro.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information about the SOM_Test, SOM_TestC, and SOM_Expect macros.

GOSK0035I \"*string***\":** *dstring***: Warning, failed -** *string***.**

Explanation: The program failed the condition specified on a SOM_TestC or SOM_Expect macro.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information about the SOM_TestC and SOM_Expect macros.

GOSK0036E Error: \"*string*\": *dstring*: Error, failed - *string*.

Explanation: The program failed the condition specified on a SOM_Test macro.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information about the SOM_Test macro.

GOSK0037I \"*string*\": *dstring*: Assertion passed - *string*.

Explanation: The program passed the condition specified on a SOM_Assert macro.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information about the SOM_Assert macro.

GOSK0038W Warning: \"*string*\": *dstring*: Warning: Assertion failed, code *dstring*-% 3d-*dstring*.

Explanation: The program failed the condition specified on a SOM_Assert macro with an error code of SOM_Warn. The message is followed by message GOSK0040I.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information about the SOM_Assert macro. See the SOM Kernel Error Codes for more information.

GOSK0039E Error: \"*string*\": *dstring*: Error: Assertion failed, code *dstring*-% 3d-*dstring*.

Explanation: The program failed the condition specified on a SOM_Assert macro with an error code other than SOM_Warn. The message is followed by message GOSK0040I.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information about the SOM_Assert macro. See the SOM Kernel Error Codes for more information.

GOSK0040I (*string*).

Explanation: This is an information message describing the condition tested by the SOM_Assert macro.

User Response: None required.

GOSK0041W Warning: Class <*string*> could not be located.

User Response: Check your program logic. Explanation: The SOM Class Manager could not locate the specified class.

GOSK0043W Warning: *string*: Method was *string*.

Explanation: An error was detected with a method. An additional error message describes the specific problem.

User Response: Correct error as directed in the following error message.

GOSK0044I {An instance of class *string* at address % 8lX

Explanation: The somPrintSelf method implemented by the SOMObject Class was invoked. Displayed is the object's class name and the address of the object in storage.

User Response: None required.

GOSK0045W Warning: somFreeObj was called, change to somFree.

Explanation: The somFreeObj function is an obsolete function.

User Response: Correct your program to call the SOMFree function instead. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for information about the SOMFree function.

GOSK0046W Warning: SOMClass::somDescendedFrom failed, self was:

Explanation: The method somDescendedFrom determined that the specified classes were not derived from each other.

User Response: Check the target class specified in message GOSK0047W and correct the error in your program logic.

GOSK0047W Warning: Target class was:

Explanation: Identifies the object that was not the ancestor of somSelf. This message follows message GOSK0046W.

User Response: Check your program logic to ensure that the ancestor class and somSelf are the intended objects.

GOSK0048I Classes were (in order) from parent of <self>:

Explanation: The somDescendedFrom method lists the parent classes of somSelf.

User Response: None required.

GOSK0049W Warning: <*string*> is not a registered class.

Explanation: The specified class has not been registered with the SOM Class Manager.

User Response: Check your program logic.

GOSK0050W Warning: <*string*> is not descended from SOMClassMgr.

Explanation: The object that the program is attempting to merge with a SOM class manager object is not a descendant.

User Response: Check your program logic to ensure the correct class manager objects were specified.

GOSK0051W Warning: \"*string*\": *dstring*: SOM Error - code = *dstring*-% 3d-*dstring*, severity = Warning.

Explanation: The default implementation of SOM_Error was invoked. The error code passed in indicated a warning level error and SOM_WarnLevel was >0.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for information about the SOM_Error macro. See the SOM Kernel Error Codes for more information.

GOSK0052I \"*string*\": *dstring*: SOM Error - code = *dstring*-% 3d-*dstring*, severity = Informational.

Explanation: The default implementation of the SOM_Error macro was invoked. The error code passed in indicated an informational level error and SOM_WarnLevel was >1.

User Response: If the result is as expected, no action is required. Otherwise, check your program logic. See *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information on the SOM_Error macro. See the SOM Kernel Error Codes for more information.

GOSK0053W Warning: \"*string*\": *dstring*: SOM Error - code = *dstring*-% 3d-*dstring*, severity = Unspecified.

Explanation: The default implementation of the SOM_Error macro was invoked. The error code passed in indicated something other than SOM_Ok, SOM_Warn, SOM_Ignore or SOM_Fatal.

User Response: Check your program logic and correct the error code passed in on the SOM_Error macro. See the SOM Kernel Error Codes for more information.

GOSK0055E Error: somDataResolve error: class <*string*> is abstract with respect to <*string*>

Explanation: SOM could not locate the requested data in the object because the class is an abstract class. It has no data.

User Response: Check your program logic

GOSK0056E Error: somDataResolve error: class <*string*> is not derived from <*string*>

Explanation: SOM could not locate the requested data because the object's class was not a subclass of the specified class.

User Response: Check your program logic.

GOSK0057E Error: class <*string*> version mismatch (version compatible with (*dstring*,*dstring*) requested by caller, but version (*dstring*,*dstring*) is provided by the class's implementation)

Explanation: The client program tried to access a class where the major version expected by the client program did not match the version of the class or the minor version expected by the client is later than that implemented by the class.

User Response: Check your program logic to ensure the client program is set up to access the intended version of the class. If the client program is correct, then check to ensure the class code being used is at the correct level.

GOSK0058E Error: Method Resolution Error: method <*string*> invoked on an object of class <*string*>.

Explanation: SOM used offset resolution to invoke the method against the object and could not locate the specified method.

User Response: Check program logic to ensure the method is supported by the class of the object.

GOSK0059I {The class \"*string*\"}

Explanation: This message is issued when the SOMClass's somPrintSelf method is invoked. It displays the class's name.

User Response: None required.

GOSK0060I {An instance of class *string* at address % 8lX

Explanation: SOMObject's default somDumpSelf method was invoked against the specified class. Additional text provides information about a class instance, for example, for an instance of the SOMClass the message provides the class name, parent name, and method information.

User Response: None required.

GOSK0061I }

Explanation: SOMobject's default somDumpSelf method was invoked. This message indicates the end of the information from the dump self.

User Response: None required.

GOSK0062E Error: somAddMethod error: can't add \"*string*\" -- no more room in mtab for \"*string*\" methods.

Explanation: During class initialization, the number of static methods exceeded the maximum allowed number.

User Response: Check to ensure the number of static methods for the class does not exceed the maximum.

GOSK0063E Error: DTS C++ class error: missing no-argument constructor called on object of class *string*.

Explanation: The DTS C++ class did not provide a no-argument constructor.

User Response: Add a no-argument constructure to the DTS C++ class.

GOSK0064E

Explanation: Fatal error occurred

User Response: Check accompanying message to correct problem

GOSK0066E DTS C++ class error: missing default memory allocator in metaclass *string*. That class did not do an override of somClassAllocate.

Explanation: A _somNew was attempted with a DTS metaclass

User Response: Update the metaclass indicated in the message to have an override of somClassAllocate.

GOSK0067E Metaclass *string* for class *string* does not exist.

Explanation: If an explicit metaclass is specified, that metaclass has to exist. SOM attempted to locate the class using somFindClass. If it was not registered with SOMClassMgr and attempt was made to load the class with information from the Interface Repository. If the DLL load failed GOSK0068W message was issued with information about the DLL and class name.

User Response: A possible cause would be that the creation of the metaclass specified failed and was not detected. Attempt to create the metaclass by invoking the metaclass newclass function directly. If you are able to create the metaclass successfully. Check the newclass function of the class indicated in the message to see if the metaclass newclass function invocation exists there.

GOSK0068W Warning: Class Initialization Error: Initialization module for class *string* missing from class library *string*"

Explanation: SOM attempted to load a class library but was either unable to locate the class library or was missing the class initialization module.

User Response: Ensure the class library dynamic link library is accessible to your program. Add the class initialization module to the class's dynamic link library.

GOSK0069E Error: The SOM Kernel currently executing is not in Link Pack Area, SOM initialization is being terminated.

Explanation: The level of SOM being executed expects the SOM Kernel(GOSSOMK) to be in Link Pack Area. If you received this message the SOM Kernel you are running is most likely in a data set that is in your steplib.

User Response: Check the data sets in your steplib concatenation and remove the one that contains GOSSOMK. If after doing this your program fails because GOSSOMK can not be located. This will indicate that SOM was not installed on your system properly. Contact your next level of support. Have them verify that SGOSLPA data set was designated as an LPA data set.

GOSK0070E Error: The initialization of the SOM Kernel in Common Storage Area has not completed, SOM initialization for this program can not complete.

Explanation: The Common Storage Area copy of the SOM Kernel does not exist. The SOM Kernel initialization in the users address space can not be completed.

User Response: Contact your next level support. Have them determine if the Distributed System Object Model (DSOM) daemon initialization is complete.

GOSK0071E Error: The System Object Model (SOM) subsystem is not active. It must be started before SOM runtime can be used.

| **Explanation:** The System Object Model (SOM) subsystem has to be active before the SOM runtime can initialize in
| any other user address space

| **User Response:** Contact your next level of support and have the SOM subsystem started.

GOSK0072I SOMIR datasets not specified result may be unpredictable.

Explanation: SOMIR not specified

User Response: Specify IR dataset desired

Naming Service Messages

Following are the texts, explanations, and user responses for the messages issued by the Naming Service. These messages appear in the program listing.

GOSN0001I Failed to create database '*string1*'. create_database failed with error code '*string2*'

Explanation: A VSAM error occurred while trying to create a database during a FENC_bind_context.

User Response: Resolve the VSAM error and try again.

GOSN0002I Name resolution failed. Unable to find '*string1*'

Explanation: Either a database could not be opened, or the name could not be found in the database.

User Response: Modify your filter string and try again.

GOSN0003I NameStack full. Reached internal limit. Stack Size: '*string1*'

Explanation: The number of elements in a naming context exceeded an internal limit during a find_any call.

User Response: Decrease the number of elements in the naming context.

GOSN0004I NameStack empty. Search will fail

Explanation: An internal error occurred during a find_any_binding function. An attempt was made to POP an empty stack.

User Response: None. Contact the IBM support center.

GOSN0009I Failed to create '*string1*'. bdb_create failed with error code '*string2*'

Explanation: a VSAM error occurred trying to create the database.

User Response: Resolve the VSAM error and try again.

GOSN0010I Failed to open '*string1*'. bdb_open failed with error code '*string2*'

Explanation: a VSAM error occurred trying to open a database.

User Response: Resolve the VSAM error and try again.

GOSN0011I Failed to destroy file '*string1*'. bdb_destroy failed with error code '*string2*'

Explanation: a VSAM error occurred during bdb_destroy.

User Response: Resolve the VSAM error and try again.

GOSN0012I Failed to replace record in file '%1$1s'. bdb_replace failed with error code '*string2*'.

Explanation: A VSAM error occurred during bdb_replace.

User Response: Resolve the VSAM error and try again.

GOSN0013I Failed to delete record in file '*string1*'. bdb_delete failed with error code '*string2*'

Explanation: A VSAM error occurred during bdb_delete.

User Response: Resolve the VSAM error and try again.

GOSN0014I Failed to decode data . tc_decode failed

Explanation: Failure occurred in a decode method.

User Response: Contact the IBM support center.

GOSN0015I Failed to encode data. tc_encode failed

Explanation: Failure occurred in an encode method.

User Response: Contact the IBM support center.

GOSN0016I Failed to close file '*string1*'. bdb_close failed with error code '*string2*'

Explanation: A VSAM error occurred during bdb_close.

User Response: Resolve the VSAM error and try again.

GOSN0017I Database handle empty. Database not open

Explanation: Internal error occurred during.

User Response: Contact the IBM support center.

GOSN0018I FilterStack full. Exceeded stack limit. Stack Limit is: '*string1*'

Explanation: A stack limit was exceeded.

User Response: If the error occurred processing a user-supplied constraint string, try shorting the sting. Otherwise, contact the IBM support center.

GOSN0019I FilterStack empty

Explanation: an attempt was made to pop an element off an empty stack.

User Response: Contact the IBM support center.

GOSN0020I NodeStack full. Exceeded stack limit . Node Stack Limit: '*string1*'. Constraint may be too long.

Explanation: a stack limit was exceeded.

User Response: If the error occurred processing a user-supplied constraint string, try shorting the sting. Otherwise, contact the IBM support center.

GOSN0021I NodeStack empty

Explanation: an attempt was made to pop an element off an empty stack.

User Response: Contact the IBM support center.

GOSN0022I Failed to parse constraint expression '*string1*'

Explanation: Constraint expression did not conform to BNF description.

User Response: Correct the expression and try again.

GOSN0024I Failed to parse constraint expression. Syntax error in '%1$s'

Explanation: Constraint expression contained an invalid syntax.

User Response: Correct the expression and try again.

GOSN0025I Root of syntax graph empty

Explanation: an internal error occurred while parsing a constraint expression.

User Response: Contact the IBM support center.

GOSN0026I Unable to build syntax graph. create_subtree failed

Explanation: Constraint expression did not conform to BNF description.

User Response: Correct the expression and try again.

GOSN0029I Failed in Array_copy

Explanation: An attempt was made to copy an array of size zero.

User Response: Contact the IBM support center.

GOSN0030I Failed in Sequence_copy

Explanation: an internal error occurred.

User Response: Contact the IBM support center.

GOSN0031I Failed to convert long

Explanation: an attempt was made to convert a "double" to a long.

User Response: Contact the IBM support center.

GOSN0032I Invalid Operands for arithmetic operations on *any*

Explanation: an attempt was made to multiply a TC value that is not a number.

User Response: Contact the IBM support center.

GOSN0034I Division by zero error

Explanation: an internal error occurred.

User Response: Contact the IBM support center.

GOSN0036I Out of memory

Explanation: "New" for a single character failed.

User Response: Free up memory and try the operation again.

Object Services (OS) Server Messages

Following are the texts, explanations, and user responses for the messages issued by the Objects Services Server. These messages appear in the program listing.

GOSO0002I Could not find implementation definition.

Explanation: The implementation definition given to SOMOSSVR could not be located in the implementation repository.

User Response: Insure that the implementation definition provided (either an implementation id or an alias name) is correct and exists in the Implementation Repository. Use the regimpl utility to list the current contents of the Implementation Repository. Insure that the implementation definition provided to SOMOSSVR (alias or id) matches the repository exactly, including upper and lower case.

GOSO0003I SOMOA_impl_is_ready failed.

Explanation: The SOMOA is not ready. Possible reasons are that the implementation repository is not found, no server class exists, or the server is disabled.

User Response: Check for the indicated reasons.

GOSO0004I Usage: somossvr [-i] <implid> | -a <alias>.

Explanation: This message indicates SOMOSSVR was invoked with an invalid argument list. One of the following was given:

- Neither an implementation id (unique class\server string) nor a "-a" followed by an implementation alias were specified, or
- Both an implementation id and an implementation alias were specified, or
- An option unknown to SOMOSSVR, such as "-x", was specified.

User Response: Correct the arguments and to SOMOSSVR. Specify either an implementation id string or "-a" alias-name but not both. The SOMOSSVR initializer argument "-i" may be omitted, precede the implementation specification, or follow it.

GOSO0005I somOS::Server (*string*) - Ended"

Explanation: The specified object services server ended operation.

User Response: None required.

GOSO0007I Cannot open filename database (<somddir>.SOMOSDB.DAT).

Explanation: The SOMOSSVR was not able to open the VSAM file "<somddir>.SOMOSDB.DAT" for I/O, where <somddir> is the high-level-qualifier specified by the SOMDDIR variable.

User Response: Ensure the SOMDDIR variable is set correctly and the server program has READ/WRITE access to any file created under the qualifier. Check the SOMERROR.LOG for more specific information (generally, an fopen() failure message).

GOSO0008I Server not initialized. Attribute persistence database *string1* cannot be accessed.

Explanation: SOMOSSVR cannot access the attribute persistence database, a VSAM file under the SOMDDIR high-level-qualifier.

User Response: Ensure the SOMDDIR variable is set correctly and the server program has READ/WRITE access to any file created under the qualifier. Check the SOMERROR.LOG for more specific information (generally, an fopen(), fread(), fwrite(), or flocate() error message).

GOSO0009I Metadata database *string1* cannot be accessed.

Explanation: SOMOSSVR cannot access the metadata database, a VSAM file under the SOMDDIR high-level-qualifier.

User Response: Ensure the SOMDDIR variable is set correctly and the server program has READ/WRITE access to any file created under the qualifier. Check the SOMERROR.LOG for more specific information (generally, an fopen(), fread(), fwrite(), or flocate() error message).

GOSO0010I Server not initialized. Attribute persistence database name not found in filename database (<somddir>.SOMOSDB.DAT).

Explanation: No attribute persistence database specific to the class(es) served was found in the global filename database. Generally, this indicates that the SOMOSSVR has not been initialized yet for the class. Other possibilities include modification of SOMOSDB.DAT outside of object-services control. SOMOSDB.DAT cannot be copied or otherwise modified outside of object services.

User Response: If this is the first invocation of SOMOSSVR for the specified implementation, include the "-i" initializer parameter so that appropriate databases are created and registered.

GOSO0011I Server not initialized. Metadata database name not found in filename database (<somddir>.SOMOSDB.DAT).

Explanation: No metadata database specific to the class(es) served was found in the global filename database. Generally, this indicates that the SOMOSSVR has not been initialized yet for the class. Other possibilities include modification of SOMOSDB.DAT outside of object-services control. SOMOSDB.DAT cannot be copied or otherwise modified outside of object services.

User Response: If this is the first invocation of SOMOSSVR for the specified implementation, include the "-i" initializer parameter so that appropriate databases are created and registered.

GOSO0012I DSOM runtime initialization failure.

Explanation: The SOMOSSVR's attempt to initialize the DSOM environment (SOMD_Init() call) failed.

User Response: Check the SOMERROR.LOG file for error indications specific to the DSOM runtime environment.

GOSO0013I somOS::Server (*string*) - Ready"

Explanation: The specified object services server is ready for operation.

User Response: Elation. ** :-) **

Event Management Framework Messages

Following are the texts, explanations, and user responses for the messages issued by the Event Management Framework.

These messages appear in the program listing.

GOSV0001E Unable to locate Sockets class *string* **in SOMIR or failed to load the associated DLL.**

Explanation: The specified sockets class could not be found.

User Response: Ensure that the correct IR is specified and that the sockets DLL is accessible.

GOSV0003E Error *dstring* **querying file descriptor** *dstring***.**

Explanation: Unable to retrieve file status information.

User Response: Ensure that a valid file descriptor was past on the sink event registration.

Security Services Messages

Following are the texts, explanations, and user responses for the messages issued by the Security Service. These messages appear in the program listing.

GOSY0152I Authentication error - Principal '*string1*' on host '*string2*' requesting service from Server '*string3*'

Explanation: client failed to acquire an authentication token from the security server.

User Response: verify that the client is passing the right USER and PASSWD to the security server. Make sure the client's machine can communicate with the security server's machine. Make sure USER and PASSWD are the ones required to authenticate the client on the security server's platform.

GOSY0153I Authentication error - Principal '*string1*' requesting service from Server '*string2*'

Explanation: This principal has failed to authenticate itself to the target server.

User Response: The client requesting service needs to verify his/her USER and PASSWD variables are correctly set in the somsec stanza of the environment variables file and that they can be used to authenticate the client on the target system. If those variables are correctly set, administrator's intervention is needed on security server's platform to verify the client is defined to the RACF database in the case of OS/390.

GOSY0154I Authentication token requested for non-secure Server '*string1*' by Principal '*string2*'

Explanation: client is requesting an authentication token for use with a target server. The security server could not compute such a token and thus assumes the target server is set to be non-SECURE in its implementation repository.

User Response: If the client is able to successfully request the service then nothing needs to be done (server is set to be non-SECURE. On the other hand if the client fails to request service then make sure the application server is registered SECURE in its implementation repository (use regimpl) then shut down the application server and restart it. After that the client needs to attempt requesting the service.

GOSY0158I inter-ORB Protocol (IOP) error - unrecognized authentication data stream.

Explanation: the security message did not carry appropriate IOP flag.

User Response: This should not happen when both the client and the server are DSOM, since the flag is hard wired into the security message and set to be of IOP type. Should this occur due to data corruption while in transport, the client is required to restart the request.

GOSY0159I Quality of Protection (QOP) error - Server '*string1*' does not support QOP requested by Principal '*string2*' on host '*string3*'.

Explanation: the security support on the target server does match what the client is requesting to use.

User Response: Normally this would not happen because the QOP values are now hard wired to be the same on both the client and the server side. This can only happen if an error occurs during the transport of a client's message to the server over the network resulting in the transported QOP being corrupted.

GOSY0161I Principal '*string1*' on host '*string2*' authenticating to non-secure server '*string3*'

Explanation: client is requesting an authentication token for use with a target server. The security server could not compute such a token and thus assumes the target server is set to be non-SECURE in its implementation repository.

User Response: If the client is able to successfully request the service then nothing needs to be done (server is set to be non-SECURE. On the other hand if the client fails to request service then make sure the application server is registered SECURE in its implementation repository (use regimpl) then shut down the application server and restart it. After that the client needs to attempt requesting the service.

GOSY0162I somsec int error - attempt to log unrecognized message, number '*string1*'

Explanation: A message number is out of range for somsec

User Response: Inform your IBM service team of the message number.

GOSY0168I Security server is out of caching-storage for keys

Explanation: The security server maintains server keys in memory and it did run out of its maximum limit. The cache needs to be cleared.

User Response: Administrator intervention is needed to shut down and restart all application servers.

GOSY0169I A reference to either the KDC or the REGISTRY object is not found

Explanation: Could not find a reference to either KDC or Registry object from the naming service

User Response: restart the som-configuration process and make sure your naming server is up.

Configuration Messages

Following are the texts, explanations, and user responses for the messages issued by the Config Framework. SOMobjects can return one of the messages described in this section in response to a user of the SOMobjects Configuration Utility.

These messages appear in the program listing.

GOSZ0001I SOM CONFIGURATION UTILITY (C) Copyright IBM Corp. 1994, 1997. All rights reserved.

Explanation: SOM Configuration Utility (SOM@CFG) copyright banner.

User Response: None

GOSZ0002I Verifying environment variables and configuration file(s) ...

Explanation: Configuration progress message. SOM@CFG is inspecting your environment variables and configuration file(s) for required settings.

User Response: None

GOSZ0004I Alias ***string*** **already exists in the Implementation Repository.**

Explanation: SOM@CFG attempted to register alias *string* in the Implementation Repository, but the alias was already registered. This can happen if you previously ran SOM@CFG or registered alias *string* manually with REGIMPL before running SOM@CFG.

User Response: Generally no response is required. However, if you changed your HOSTNAME or SOMDPORT values in your configuration file since the last time you ran SOM@CFG, you must delete the Implementation Repository data sets and re-run SOM@CFG. The Implementation Repository data sets are:

1. SOMMVS.ALIAS.DB
2. SOMMVS.IMPL.DB
3. SOMMVS.ALIASDAT.DB

GOSZ0005I Registering alias ***string*** **in the Implementation Repository.**

Explanation: Configuration progress message. SOM@CFG is now registering alias *string* in the Implementation Repository.

User Response: None.

GOSZ0006I Updating SOMNMOBJREF variable in configuration file: ***string*****.**

Explanation: Configuration progress message. SOM@CFG is now writing the SOMNMOBJREF variable setting to the configuration file. The value of SOMNMOBJREF is set to the stringified value of the local (root) Extended Naming Context (ENC) object, which resides within local naming server process.

User Response: None.

GOSZ0007I Configuration has successfully completed.

Explanation: Configuration progress message. SOM@CFG has successfully completed the configuration process.

User Response: None.

GOSZ0008I Creating object of class ***string*****.**

Explanation: Configuration progress message. SOM@CFG is now creating an instance of class *string.*

User Response: None.

GOSZ0009I Initializing the database for *string* server.

Explanation: Configuration progress message. SOM@CFG is now creating somOS::Server databases for the *string* server.

User Response: None.

GOSZ0010I Configuration file variable *string.* in stanza *string* is not set. Using default: *string.*.

Explanation: Configuration file variable *string* has been defaulted to value *string.*.

User Response: Generally none. If the indicated default is not correct for your installation, terminate SOM@CFG, update its configuration file, then restart SOM@CFG.

GOSZ0011I Building global naming tree ...

Explanation: Configuration progress message. SOM@CFG is now creating the objects that comprise the global naming tree.

User Response: None.

GOSZ0012I Starting configuration process ...

Explanation: Configuration progress message. SOM@CFG is now starting the actual configuration process.

User Response: None.

GOSZ0014I Completing configuration for the *string* server.

Explanation: Configuration progress message. SOM@CFG is now entering the final phase of configuration for the *string* server.

User Response: None.

GOSZ0018I Configuration has been user-terminated prior to completion.

Explanation: SOM@CFG was terminated by the user prior to completion.

User Response: Re-run SOM@CFG when you are ready.

GOSZ0019I som@cfg configures the Naming, Security, and LifeCycle services.

Explanation: Part of SOM@CFG's help messages. Accompanied by GOSZ0020I when you issue SOM@CFG with either no parameters or an unrecognized parameter.

User Response: None.

GOSZ0020I Usage: som@cfg [-i | -d]

```
          -i to configure system as install host

          -d to configure system as DSOM host

     som@cfg uses the file specified on the SOMENV environment
     variable or the SOMENV DD.
```

Explanation: Part of SOM@CFG's help messages. Accompanied by GOSZ0019I when you issue SOM@CFG with either no parameters or an unrecognized parameter.

User Response: None.

GOSZ0028E ERROR: Configuration variable *string.* in stanza *string* is not set.

Explanation: Required configuration file variable *string* is not set.

User Response: Set variable, then re-run SOM@CFG.

GOSZ0029E ERROR: Unable to write to file: ***string*****.**

Explanation: SOM@CFG failed to write to file *string* successfully. The configuration process has been terminated.

User Response: Correct cause for write failure, then re-run SOM@CFG. Likely causes for write failure:

- File (data set) name misspelled. If file located in HFS, remember that HFS file names are case-sensitive.
- Fully qualified MVS dataset name must be quoted.
- User identity associated with SOM@CFG does not have write access (e.g. via RACF) to the file.
- Another job (or user) has the file allocated.
- If the file is located in the HFS, the write permission bit may not be set.

GOSZ0030E ERROR: This system is already configured as ***string*** **host.**
If you want to reconfigure this system, you need to remove variables HOSTKIND and SOMNMOBJREF from stanza [somnm].

Explanation: SOM@CFG uses the presence of the HOSTKIND and SOMNMOBJREF variables to determine whether or not SOMobjects has been configured on this system. If a previous configuration is detected, the configuration process is terminated.

User Response: If you are trying to re-configure SOMobjects, follow the instructions in the message, then re-run SOM@CFG.

GOSZ0032E ERROR: Unable to write to any files listed in environment variable SOMENV.

Explanation: The first (or only) configuration file specified on the SOMENV environment variable could not be opened for write access. The configuration process has been terminated.

User Response: Correct reason for write failure, then re-run SOM@CFG. Likely causes for write-failure:

- File (data set) name misspelled. If file located in HFS, remember that HFS file names are case-sensitive.
- Fully qualified MVS dataset name must be quoted.
- User identity associated with SOM@CFG does not have write access (e.g. via RACF) to the file.
- Another job (or user) has the file allocated.
- If the file is located in the HFS, the write permission bit may not be set.

GOSZ0034E ERROR: Unable to read from file: ***string*****.**

Explanation: SOM@CFG was unable to read from file *string.* The configuration process has been terminated.

User Response: Correct reason for read failure, then re-run SOM@CFG. Likely causes for read-failure:

- File (data set) name misspelled. If file located in HFS, remember that HFS file names are case-sensitive.
- Fully qualified MVS dataset name must be quoted.
- User identity associated with SOM@CFG does not have read access (e.g. via RACF) to the file.
- Another job (or user) has the file allocated DISP=OLD.
- If the file is located in the HFS, the read permission bit may not be set.

GOSZ0036E ERROR: Error manipulating an LNameComponent object.
Exception name: [***string*****]**
Error code: [***dstring*****].**

Explanation: A fatal error was encountered while invoking a method on the indicated object. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0037E ERROR: Error manipulating an LName object.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: A fatal error was encountered while invoking a method on the indicated object. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0038E ERROR: Cannot instantiate an object of class *string*.

Explanation: The attempt to create the indicated object failed. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0040E ERROR: Unable to allocate storage.

Explanation: SOM@CFG ran out of HEAP space.

User Response: Increase the HEAP runtime option setting and re-run SOM@CFG.

GOSZ0041E ERROR: Cannot find implementation definition for *string*.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: The implementation repository does not contain the indicated class definition. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0042E ERROR: Cannot find *string* Server reference implementation def.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: SOM@CFG could not locate the implementation repository definition for the indicated server. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0043E ERROR: Cannot get *string* Server object from implementation def.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: SOM@CFG could not create a server proxy from the implementation repository definition for the indicated server. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0044E ERROR: Cannot create *string* object.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: SOM@CFG failed to create the indicated object. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0045E ERROR: Failed to bind *string* object.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: SOM@CFG failed to bind the indicated object in naming tree. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0046E ERROR: Environment variable *string* must be set.

Explanation: The indicated environment variable is required, but was not set. The configuration process has been terminated.

User Response: Set environment variable, then re-run SOM@CFg.

GOSZ0048E ERROR: Error initializing server *string*.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: The initialization of the indicated server failed. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0050E ERROR: This system is improperly configured.
If you want to reconfigure this system, you need to remove variable SOMNMOBJREF from stanza [somnm].

Explanation: SOM@CFG uses the presence of the HOSTKIND and SOMNMOBJREF variables to determine whether or not SOMobjects has been configured on this system. If a previous configuration is detected, the configuration process is terminated.

User Response: If you are trying to re-configure SOMobjects, follow the instructions in the message, then re-run SOM@CFG.

GOSZ0052E ERROR: Invalid format for GLOBAL_OBJREF_FILE: *string*.

Explanation: The indicated file contains corrupted or incorrect information. Note this problem only affects a system installing as a DSOM host (i.e. the -d option was specified on SOM@CFG).

User Response: Ensure the GLOBAL_OBJREF_FILE configuration variable correctly specifies the same file that was specified for the system that was configured as the install host (i.e. was configured with the "-I" option on SOM@CFG), then re-run SOM@CFG.

GOSZ0054E ERROR: Unable to bind factory context *string*.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: The factory ENC could not be registered with naming server. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0056E ERROR: Failed to create global naming tree.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: Global naming tree creation failed. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0057E ERROR: Unable to initialize DSOM. SOMD_Init failed.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: Initialization for distributed SOMobjects failed. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0058E ERROR: Failed to bind global root naming context to local root naming context.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: The global ENC could not be bound to the local ENC. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0059E ERROR: Failed to add alias *string* in the Implementation Repository.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: The indicated alias could not be added to the implementation repository. This error may be accompanied by a GOSBxxxx message, which provides additional information. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0060E ERROR: Unable to convert FileXNaming::FileENC object
to its string representation.
Exception name: [*string*]
Error code: [*dstring*].

Explanation: The object reference of the indicated object could not be converted to string form. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0061E ERROR: Unable to convert Global root naming context string to its FileXNaming::FileENC object.
Exception name: [*string*]
Error code: [*dstring*]."

Explanation: The stringified object reference for the indicated object could not be converted back to object form. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0063E ERROR: Unable to disable authentication.

Explanation: Configuration could not disable authentication, which is required for the configuration process. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0065E ERROR: Unable to resolve method *string* of class *string*.

Explanation: The indicated method could not be resolved on the indicated class. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0066E ERROR: Unable to execute method *string* of class *string*.
Exception name: [*string*]
Error code: [*dstring*]."

Explanation: The indicated method failed execution. The configuration process has been terminated.

User Response: Report problem to IBM.

GOSZ0067E ERROR: Unable to find class *string* in the Interface Repository. Ensure that the SOMIR environment variable is specified."

Explanation: The interface repository does not contain the indicated class definition. The configuration process has been terminated.

User Response: Ensure the SOMIR= configuration file variable includes the IBM-supplied interface repository files in its list, then re-run SOM@CFG. See SOMMVS.SGOSPROF(GOSENV1) for reference. If that does not fix the problem, report it to IBM.

SOMRAS Framework Messages

Following are the texts, explanations, and user responses for the messages issued by the SOMRAS Framework. SOMobjects can return one of the messages described in this section in response to a user of the SOMRAS Framework.

These messages appear in the program listing.

The suffix of each message is a type code that can be one of the following:

I Informational

W Warning

E Error

GOS20001I Raised *string1 string2* **with severity** *string3* **at** *string4***:***string5***. Request from** *string6* **on** *string7***. Error code is** *string8***[***string9***].** *string10*

Explanation: The system intercepted a USER or SYSTEM exception. The following describes each parameter:

string1	The exception type, either USER or SYSTEM.
string2	The class/method which detected the exception.
string3	The severity: INFO, WARNING, ERROR, or MAPPED_EXCEPTION.
string4	The detecting module.
string5	The __LINE__ within the detecting module.
string6	The client which originated the request, if any.
string7	The host name of the server of the request.
string8	The error code.
string9	The text form of the error.
string10	An optional string providing additional diagnostics data (if any). If this string is non-null, see the message ID associated with that text.

User Response: see Chapter 7, "SOMobjects Error Codes" on page 7-1 for the appropriate response for the specified error code that is identified in string8. Additionally, if string10 is non-null, see the message ID identified in that string.

GOS20002I Mapped *string1 string2* **to** *string3 string4* **at** *string5***:***string6***. Request from** *string7* **on** *string8***. New Error code is** *string9***[***string10***].** *string11*

Explanation: The system intercepted a USER or SYSTEM exception. The following describes each parameter:

string1	The exception type, either USER or SYSTEM.
string2	The class/method which detected the exception.
string3	The severity: INFO, WARNING, ERROR, or MAPPED_EXCEPTION.
string4	Indicates the "mapped to" exception name.
string5	The detecting module.
string6	The __LINE__ within the detecting module.
string7	The client which originated the request, if any.
string8	The host name of the server of the request.
string9	The error code.
string10	The text form of the error.
string11	An optional string providing additional diagnostics data (if any). If this string is non-null, see the message ID associated with that text.

User Response: see Chapter 7, "SOMobjects Error Codes" on page 7-1 for the appropriate response for the specified error code that is identified in string9. Additionally, if string11 is non-null, see the message ID identified in that string.

GOS20003I Process abnormally terminated at *string1*:*string2*. Request from *string3* on *string4*. Error code is *string5*[*string6*]. *string7*

Explanation: The system detected a condition which caused an abnormal termination. The following describes each parameter:

string1	The detecting module.
string2	The __LINE__ within the detecting module.
string3	The client which originated the request, if any.
string4	The host name of the server of the request.
string5	The error code.
string6	The text form of the error.
string7	An optional string providing additional diagnostics data (if any). If this string is non-null, see the message ID associated with that text.

User Response: see Chapter 7, "SOMobjects Error Codes" on page 7-1 for the appropriate response for the specified error code that is identified in string5. Additionally, if string7 is non-null, see the message ID identified in that string.

GOS20004I Service data collected with severity *string1* at *string2*:*string3*. Request from *string4* on *string5*. Error code is *string6*[*string7*]. *string8*

Explanation:

string1	The severity of the error: INFO, WARNING, ERROR, or MAPPED_EXCEPTION.
string2	The detecting module.
string3	The __LINE__ within the detecting module.
string4	The client which originated the request, if any.
string5	The host name of the server of the request.
string6	The error code.
string7	The text form of the error.
string8	An optional string providing additional diagnostics data (if any). If this string is non-null, see the message ID associated with that text.

User Response: see Chapter 7, "SOMobjects Error Codes" on page 7-1 for the appropriate response for the specified error code that is identified in string6. Additionally, if string8 is non-null, see the message ID identified in that string.

GOS20007I Trace data set is *string*.

Explanation: The specified name is the name of the trace data set.

User Response: None required.

GOS20008I Trace initialization failure. Error code: *dstring* Reason: *dstring* Extended code: *dstring*

Explanation: There was an error in the allocation/creation of the trace dataset or trace facilities.

User Response: Contact the IBM support center.

GOS20009I No Trace data set will be used.

Explanation: No trace data set is being used.

User Response: None required.

GOS20010I Tracing has ended.

Explanation: Tracing has ended.

User Response: None required.

GOS20011I Error occurred while writing trace record. Error code: *dstring* **Reason:** *dstring*

Explanation: An error in one of the calls to the internal trace routines.

User Response: Contact the IBM support center.

GOS20012I Error occurred while terminating trace. Error code: *dstring* **Reason:** *dstring*

Explanation: An error in one of the calls to the internal trace routines.

User Response: Contact the IBM support center.

GOS20013I MVSTraceLog value *string***, does not contain a valid MVS qualifier.**

Explanation: The MVSTraceLog value was incorrectly specified. A valid MVS High Level Qualifier must be used.

User Response: Correct the MVSTraceLog value.

GOS20014I Trace data set size will be: *dstring string***.**

Explanation: The trace data set was allocated with the size shown. This may be different from the requested amount if the requested amount was either in error or unavailable.

User Response: None required.

SOM Collection Classes Messages

Following are the texts, explanations, and user responses for the messages issued by the SOM Collection Classes. SOMobjects can return one of the messages described in this section in response to a user of the SOM Collection Classes.

These messages appear in the program listing.

GOS40001I You need to override the somfIsEqual method

Explanation: somIsEqual method is not overridden.

User Response: Override somIsEqual method.

GOS40002I (and probably override the somfHash method too)

Explanation: somfHash method is not overridden.

User Response: Override somfHash method.

GOS40003I You need to override the somfIsGreaterThan method

Explanation: somfIsGreaterThan method is not overridden.

User Response: Override somfIsGreaterThan method.

GOS40004I You need to override the somfIsLessThan method

Explanation: somfIsLessThan method is not overridden.

User Response: Override somfIsLessThan method.

GOS40005I FATAL ERROR: somf_TCollection::somfAdd needs to be overridden.

Explanation: somfAdd method is not overridden.

User Response: Override somfAdd method.

GOS40006I FATAL ERROR: somf_TCollection::somfRemove needs to be overridden.

Explanation: somfRemove method is not overridden.

User Response: Override somfRemove method.

GOS40007I FATAL ERROR: somf_TCollection::somfRemoveAll needs to be overridden.

Explanation: somfRemoveAll method is not overridden.

User Response: Override somfRemoveAll method.

GOS40008I FATAL ERROR: somf_TCollection::somfDeleteAll needs to be overridden.

Explanation: somfDeleteAll method is not overridden.

User Response: Override somfDeleteAll method.

GOS40009I FATAL ERROR: somf_TCollection::somfCount needs to be overridden.

Explanation: somfCount method is not overridden.

User Response: Override somfCount method.

GOS40010I FATAL ERROR: somf_TCollection::somfCreateIterator needs to be overridden.

Explanation: somfCreateIterator method is not overridden.

User Response: Override somfCreateIterator.

GOS40011I No Deque Specified... This behavior no longer supported

Explanation: No deque specified.

User Response: No longer supported.

GOS40012I You must use somfFirst before calling somfNext

Explanation: somfFirst is called before somfNext.

User Response: Calling somfFirst before somfNext.

GOS40013I somf_TDeque and somf_TDequeIterator are out of sync

Explanation: somf_TDeque and somf_TDequeIterator out of sync.

User Response: Re-sync them.

GOS40014I somf_THashTable and somf_THashTableIterator are out of sync

Explanation: somf_THashTable and somf_THashTableIterator out of sync.

User Response: Re-sync them.

GOS40015I FATAL ERROR: somf_Iterator::somfNext needs to be overridden.

Explanation: somfNext method is not overridden.

User Response: Override somfNext method.

GOS40016I FATAL ERROR: somf_Iterator::somfFirst needs to be overridden.

Explanation: somfFirst method is not overridden.

User Response: Override somfFirst method.

GOS40017I FATAL ERROR: somf_Iterator::somfRemove needs to be overridden.

Explanation: somfRemove method is not overridden.

User Response: Override somfRemove method.

GOS40018I switching directions not supported yet.

Explanation: Switching directions not supported.

User Response: Not supported.

GOS40019I somf_TSortedSequence and somf_TSortedSequenceIterator are out of sync

Explanation: somf_TSortedSequence and somf_TSortedSequenceIterator are out of sync.

User Response: Re-sync them.

GOS40020I You must use somfFirst or somfLast before calling somfRemove.

Explanation: somfFirst or somfLast is called before somfRemove.

User Response: Calling somfFirst or somfLast before somfRemove.

GOS40021I FATAL ERROR: somf_TSequence::somfAfter needs to be overridden.

Explanation: somfAfter method is not overridden.

User Response: Override somfAfter method.

GOS40022I FATAL ERROR: somf_TSequence::somfBefore needs to be overridden.

Explanation: somfBefore method is not overridden.

User Response: Override somfBefore method.

GOS40023I FATAL ERROR: somf_TSequence::somfLast needs to be overridden.

Explanation: somfLast method is not overridden.

User Response: Override somfLast method.

GOS40024I FATAL ERROR: somf_TSequence::somfFirst needs to be overridden.

Explanation: somfFirst method is not overridden.

User Response: Override somfFirst method.

GOS40025I FATAL ERROR: somf_TSequence::somfAdd needs to be overridden.

Explanation: somfAdd method is not overridden.

User Response: Override somfAdd method.

GOS40026I FATAL ERROR: somf_TSequence::somfRemove needs to be overridden.

Explanation: somfRemove method is not overridden.

User Response: Override somfRemove method.

GOS40027I FATAL ERROR: somf_TSequence::somfRemoveAll needs to be overridden.

Explanation: somfRemoveAll method is not overridden.

User Response: Override somfRemoveAll method.

GOS40028I FATAL ERROR: somf_TSequence::somfDeleteAll needs to be overridden.

Explanation: somfDeleteAll method is not overridden.

User Response: Override somfDeleteAll method.

GOS40029I FATAL ERROR: somf_TSequence::somfCount needs to be overridden.

Explanation: somfCount method is not overridden.

User Response: Override somfCount method.

GOS40030I FATAL ERROR: somf_TSequence::somfCreateIterator needs to be overridden.

Explanation: somfCreateIterator method is not overridden.

User Response: Override somfCreateIterator method.

GOS40031I FATAL ERROR: somf_TSequenceIterator::somfLast needs to be overridden.

Explanation: somfLast method is not overridden.

User Response: Override somfLast method.

GOS40032I FATAL ERROR: somf_TSequenceIterator::somfPrevious needs to be overridden.

Explanation: somfPrevious method is not overridden.

User Response: Override somfPrevious method.

GOS40033I FATAL ERROR: somf_TSequenceIterator::somfFirst needs to be overridden.

Explanation: somfFirst method is not overridden.

User Response: Override somfFirst method.

GOS40034I FATAL ERROR: somf_TSequenceIterator::somfNext needs to be overridden.

Explanation: somfNext method is not overridden.

User Response: Override somfNext method.

GOS40035I FATAL ERROR: somf_TSequenceIterator::somfRemove needs to be overridden.

Explanation: somfRemove method is not overridden.

User Response: Override somfRemove method.

GOS40036I You must use somfFirst before calling somfPrevious

Explanation: somfFirst is not called before somfPrevious.

User Response: Call somfFirst before somfPrevious.

GOS40037I somf_TSet and somf_TSetIterator are out of sync

Explanation: somf_TSet and somf_TSetIterator are out of sync.

User Response: Re-sync them.

GOS40038I FATAL ERROR: somf_TIndexedSequence::somfLowBound needs to be overridden.

Explanation: somfLowBound method is not overridden.

User Response: Override somfLowBound method.

GOS40039I FATAL ERROR: somf_TIndexedSequence::somfHighBound needs to be overridden.

Explanation: somfHighBound method is not overridden.

User Response: Override somfHighBound method.

GOS40040I FATAL ERROR: somf_TIndexedSequence::somfAt needs to be overridden.

Explanation: somfAt method is not overridden.

User Response: Override somfAt method.

GOS40041I FATAL ERROR: somf_TIndexedSequence::somfAtInsert needs to be overridden.

Explanation: somfAtInsert method is not overridden.

User Response: Override somfAtInsert method.

GOS40042I FATAL ERROR: somf_TIndexedSequence::somfAtPut needs to be overridden.

Explanation: somfAtPut method is not overridden.

User Response: Override somfAtPut method.

GOS40043I FATAL ERROR: somf_TIndexedSequence::somfFind needs to be overridden.

Explanation: somfFind method is not overridden.

User Response: Override somfFind method.

GOS40044I FATAL ERROR: somf_TIndexedSequence::somfAdd needs to be overridden.

Explanation: somfAdd method is not overridden.

User Response: Override somfAdd method.

GOS40045I FATAL ERROR: somf_TIndexedSequence::somfCreateIterator needs to be overridden.

Explanation: somfCreateIterator method is not overridden.

User Response: Override somfCreateIterator method.

Chapter 6. 6C4 System Abend

This chapter documents Diagnosis, Modification or Tuning Information.

The following is the description of the 6C4 OS/390 SOMobjects system abend with its error codes and associated explanations and responses.

6C4

Explanation: A reason code further describes the error:

Reason Code (hex)	**Explanation**
01060004	The SOMobjects subsystem could not be initialized because one of the following occurred: • Another SOMobjects subsystem is already active • Another SOMobjects subsystem ended without cleaning up its resources. **System Action:** The SOMobjects subsystem is not initialized. The system abnormally ends the command. The system continues other processing. **System Programmer Response:** Stop the SOMobjects subsystem that is active before starting this SOMobjects subsystem. If the subsystem is already stopped and the problem persists, start the new SOMobjects subsystem with the same name as the SOMobjects subsystem that previously stopped. If the problem persists, contact the IBM Support Center. For more information about starting and stopping a SOMobjects subsystem, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.
010F0002	SOMobjects subsystem could not open the data control block (DCB) for the C runtime library for the SOMobjects subsystem. The SOMobjects subsystem might not be installed properly. **System Action:** The system ends the application. **System Programmer Response:** Ensure that the SOMobjects subsystem is properly installed. Ensure that the GOSRTL1 data sets are correct and accessible to the userid associated with the SOMobjects subsystem. If the problem persists, contact the IBM Support Center.
02020001	The caller of a service is not in supervisor state. **System Action:** The application is abnormally ended. **System Programmer Response:** Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.
02020002	Common area storage could not be obtained. **System Action:** The system issues abend `X'6C4'` and ends the command processing. **Programmer Response:** Contact the system programmer. **System Programmer Response:** Increase the amount of common storage specified on the CSA parameter in IEASYSxx parmlib member. Re-IPL the system to activate the change. If the problem persists, contact the IBM Support Center.
02020003	An asynchronous cross memory post failed. A SOM command might have been entered while the system was processing a command to stop the SOMobjects subsystem.

System Action: The system abnormally ends the command.

Operator Response: Start SOM again. Once SOM is started, enter the SOM command again.

System Programmer Response: Restart the SOMobjects subsystem. If the problem persists, contact the IBM Support Center.

02080002 The system could not obtain common storage.

System Action: The system abnormally ends the command.

System Programmer Response: Increase the amount of common storage specified on the CSA parameter in IEASYSxx parmlib member. Re-IPL the system to activate the change. If the problem persists, contact the IBM Support Center.

02080003 An asynchronous cross memory post failed. A SOM command might have been entered while the system was processing a STOP SOM command.

System Action: The system abnormally ends the command.

Operator Response: Enter a START SOM command to start SOM again.

System Programmer Response: If the problem persists, contact the IBM Support Center.

05000001 Unsupported SOMobjects subsystem service.

System Action: The system abnormally ends the application.

User Response: Contact the system programmer.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05010001 Input to a service routine is missing. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05010002 The caller of a SOM service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05010004 The system could not obtain storage required for the server.

System Action: The system records the error and continues processing. The server does not connect to WLM.

System Programmer Response: Obtain the return code from the STORAGE obtain macro in register 0. If the problem persists, contact the IBM Support Center.

0501000A The caller is not authorized to invoke the requested function. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

0501000C The server could not disconnect from WLM.

System Action: The system records the error and continues processing.

System Programmer Response: Obtain the reason code from the IWMDISC macro in register 0. If the problem persists, contact the IBM Support Center.

05020007 The caller of a service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05030001 Input to a service routine is missing. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05030003 The caller passed a parameter list that is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05030004 The caller requested a security function that is not valid. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

0503000A The caller is not authorized to invoke the requested function. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05030109 The system could not process a RACROUTE REQUEST=AUTH request when checking command authorization to a server.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

0503000B A PC routine is not available.

System Action: The system issues abend `X'6C4'` to the caller of the SOM service. The system might issue message GOS009I.

Operator Response: Enter a START SOM command to start SOM again. If SOM is already started and the system issued message GOS009I, contact the system programmer.

System Programmer Response: If you cannot start SOM, contact the IBM Support Center.

05040001 Input to service routine missing. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05040003 The parameter list that was passed to a service routine is not valid because the caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

05040004 Function requested from service routine is not valid. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

0504000A The caller is not authorized to invoke the requested function. The caller of a service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

0504000B SOMobjects subsystem services not available.

System Action: The system issues abend `X'6C4'` to the caller of the SOM service. The system might issue message GOS009I.

Operator Response: Enter a START SOM command to start SOM again. If SOM is already started and the system issued message GOS009I, contact the system programmer.

System Programmer Response: If you cannot start SOM, contact the IBM Support Center.

05040100 During security initialization, the system could not process a RACROUTE REQUEST=LIST,ENVIR=CREATE request for the SOMDOBJS RACF security class.

System Action: The system abnormally ends the application.

System Programmer Response: Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.

05040101 During security initialization, the system could not process a RACROUTE REQUEST=LIST,ENVIR=CREATE request for the CBIND RACF security class.

System Action: The system abnormally ends the application.

System Programmer Response: Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.

05040102 During client authentication, the system could not process a RACROUTE REQUEST=LIST,ENVIR=CREATE request.

System Action: The system abnormally ends the application.

System Programmer Response: Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.

05040103 During client authentication, the system could not process a RACROUTE REQUEST=FASTAUTH request.

System Action: The system abnormally ends the application.

System Programmer Response: Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.

05040104	During client authentication or client/server disconnect, the system could not process a RACROUTE REQUEST=VERIFY ENVIR=DELETE request. **System Action:** The system abnormally ends the application. **System Programmer Response:** Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.
05040105	During security cleanup processing, the system could not process a RACROUTE REQUEST=LIST ENVIR=DELETE request. **System Action:** The system abnormally ends the application. **System Programmer Response:** Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.
05040106	During security cleanup processing, the system could not process a RACROUTE REQUEST=LIST ENVIR=DELETE request for the CBIND RACF security class. **System Action:** The system abnormally ends the application. **System Programmer Response:** Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.
05040107	During client/server connect processing, the system could not process a RACROUTE REQUEST=LIST REQUEST=VERIFY ENVIR=CREATE PASSCHK=NO request. **System Action:** The system abnormally ends the application. **System Programmer Response:** Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.
05040108	During method level authorization processing, the system could not process a RACROUTE REQUEST=FASTAUTH request. **System Action:** The system abnormally ends the application. **System Programmer Response:** Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.
05040109	During server authorization processing, the system could not process a RACROUTE REQUEST=AUTH request. **System Action:** The system abnormally ends the application. **System Programmer Response:** Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.

0504010A During server authorization or client authentication processing, the system could not process a RACROUTE REQUEST=EXTRACT,TYPE=ENCRYPT request.

System Action: The system abnormally ends the application.

System Programmer Response: Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.

0504010B During method level authorization processing, a bad parameter was passed to the service. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Register 0 contains diagnostic information in the form X'00WWYYZZ', where X'WW' is the SAF return code, X'YY' is the RACF return code, and X'ZZ' is the RACF reason code. See *OS/390 Security Server External Security Interface (RACROUTE) Macro Reference* for a description of the return and reason codes from RACROUTE.

0507000A The caller is not authorized to invoke the requested function. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

050B0001 Input to a service routine is missing. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

050B0002 The caller of a SOM service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

050C0001 Input to service routine missing. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

050C0003 The parameter list that was passed to a service routine is not valid because the caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

050C0004 Function requested from service routine is not valid. The caller of the service is not valid.

System Action: The system abnormally ends the application.

System Programmer Response: Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

050C0005 Unable to create SOM latch

System Action: The system abnormally ends the application.

System Programmer Response: Obtain the return code from the latch obtain service (ISGLCRT) in register zero. See *OS/390 MVS Auth Assembler Services Reference ENF-ITT* for descriptions of return codes from ISGLCRT.

050C0006	Unable to obtain SOM latch **System Action:** The system abnormally ends the application. **System Programmer Response:** Obtain the return code from the latch obtain service (ISGLOBT) in register zero. See *OS/390 MVS Auth Assembler Services Reference ENF-ITT* for descriptions of return codes from ISGLOBT.
050C0007	Unable to release SOM latch **System Action:** The system abnormally ends the application. **System Programmer Response:** Obtain the return code from the latch release service (ISGLREL) in register zero. See *OS/390 MVS Auth Assembler Services Reference ENF-ITT* for descriptions of return codes from ISGLREL.
050D0002	The caller of a SOM is not valid. **System Action:** The system abnormally ends the application. **System Programmer Response:** Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.
050D000B	Internal SOM error. **System Action:** The system abnormally ends the application. **System Programmer Response:** Contact the IBM Support Center.
050D00xx	Input to a SOM service routine is missing or invalid. The caller of the service is not valid. **System Action:** The system abnormally ends the application. **System Programmer Response:** Verify that the application is using standard interfaces. If the problem persists, contact the IBM Support Center.

Chapter 7. SOMobjects Error Codes

This chapter documents Diagnosis, Modification or Tuning Information.

This chapter provides information about error codes generated from SOMobjects. These error codes pertain to SOM and DSOM as well as Object Services. Informational-only messages (as opposed to error messages) are not documented.

The following are the error codes covered in this chapter.

Special Error Codes

The following error codes can be used by any Object Service:

00000 UNDEFINED

Explanation: This error code has no specific meaning. Any object service can use this error code if there is not a service-specific error code that would provide additional information. If an object service makes an error log entry to save data for service personnel, but none of the data is likely to be useful, this error code is used.

Programmer Response: None.

00001 EXCEPTION_MAPPED

Explanation: An object service received a user-exception from a sub-service method call. When the receiving object service does not list this exception on its own **raises** keyword, it must either map the received exception into the user-exception that is listed on its **raises** keyword, or a system exception. The object service can then raise the new, mapped exception to its caller.

Programmer Response: Determine the original exception that was raised. Review the error log entries or messages for a corresponding message that contains the text . . . Raised USER_EXCEPTION . . . Once you locate this message, follow the action list in this appendix for the error code it contains. This error code is usually the best place to look to find information for resolving the problem.

SOM Kernel Error Codes

You might encounter the following error codes from the SOM kernel and the various frameworks of SOMobjects while an application is running.

20011 SOMERROR_CCNullClass

Explanation: A null class argument is being passed to a **somDescendedFrom** method..

20029 SOMERROR_SompntOverflow

Explanation: The internal buffer used in the **somPrintf** function. is overflowing.

20039 SOMERROR_MethodNotFound

Explanation: **somFindMethod(Ok)** methods. is failing to find the indicated method.

20049 SOMERROR_StaticMethodTableOverflow

Explanation: A method-table overflow is occurring in **somAddStaticMethod**.

20059 SOMERROR_DefaultMethod

Explanation: A defined method was not added before calling the **somDefaultMethod**.

20069 SOMERROR_MissingMethod

Explanation: The specified method is not defined on the target object.

20079 SOMERROR_BadVersion

Explanation: An attempt is being made to load, create, or use a version of a class-object implementation that is incompatible with the program.

20089 SOMERROR_NullId

Explanation: The **som_CheckId** is being given a NULL ID to check.

20099 SOMERROR_OutOfMemory

Explanation: Memory is exhausted.

20109 SOMERROR_TestObjectFailure

Explanation: The **somObjectTest** is finding problems with the object it is testing.

20119 SOMERROR_FailedTest

Explanation: The **somTest** is detecting a failure.

20121 SOMERROR_ClassNotFound

Explanation: The **somFindClass** method. cannot find the requested class.

20131 SOMERROR_OldMethod

Explanation: An old-style method name is being used.

Programmer Response: Change the method to the appropriate name.

20149 SOMERROR_CouldNotStartup

Explanation: The **somEnvironmentNew** function. is failing to complete.

20159 SOMERROR_NotRegistered

Explanation: The **somUnloadClassFile** method. argument is not a registered class.

20169 SOMERROR_BadOverride

Explanation: The **somOverrideSMethod** is being invoked for a method that is not defined in a parent class.

20179 SOMERROR_NotImplementedYet

Explanation: The method raising the error message is not implemented.

20189 SOMERROR_MustOverride

Explanation: The method raising the error message should be overridden.

20199 SOMERROR_BadArgument

Explanation: An argument to a core SOM method is failing a validity test.

20219 SOMERROR_NoParentClass

Explanation: While creating a class object, the parent class cannot be found.

20229 SOMERROR_NoMetaClass

Explanation: While creating a class object, the metaclass object cannot be found.

DSOM Error Codes

You might encounter the following error codes from DSOM while an application is running. Obsolete messages have been removed, so message numbers are not in sequential order.

30001 SOMDERROR_NoMemory

Explanation: DSOM run time is failing to allocate the necessary memory.

Programmer Response: Bring down the DSOM application and free system resources.

30002 SOMDERROR_NotImplemented

Explanation: The function or method is not yet supported.

Programmer Response: Do not call the function or method in the application.

30003 SOMDERROR_InvalidProtocolInformation

Explanation: Invalid protocol information is being specified in the configuration file.

Programmer Response: For a description of **SOMDPROTOCOLS** and stanza definitions for each protocol, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

30004 SOMDERROR_SOMDDAlreadyRunning

Explanation: The DSOM daemon is already started. That is, an instance of **somdd** is already running.

Programmer Response: If the current instance of **somdd** is not responding properly, delete all instances of **somdd** and restart a new copy.

30006 SOMDERROR_InvalidConfigSetting

Explanation: The configuration variable setting is not valid.

Programmer Response: Use the **somdchk** command to display the current environment. Set the environment variable to a valid value. For a description of the environment variable, see *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

30007 SOMDERROR_BadEnvironment

Explanation: One of the following applies:

- An environment is being passed to a DSOM function or method and the major field is not set to NO_EXCEPTION, or
- An invalid environment structure is being returned for a user exception.

Programmer Response: If the first scenario in the Explanation applies, verify that a valid environment is being passed to DSOM. A valid environment has the major field set to NO_EXCEPTION. If the second scenario in the Explanation applies, use **somSetException** to set the exception value in the Environment structure.

30008 SOMDERROR_HostAddress

Explanation: DSOM run time is encountering an invalid host address.

Programmer Response: Verify the network setup.

1. Ensure that the HOSTNAME environment variable in the configuration file specifies the correct host name.
2. Verify that the network database that associates host names and addresses contains the correct entries for client and server machines.

30009 SOMDERROR_CouldNotStartProcess

Explanation: The DSOM daemon cannot start a new process to execute the server program. The procedure name used to start a server is specified when the application environment for the server is defined to WLM.

Programmer Response: Ensure that an application environment for the server is defined to WLM. If there is an application environment for the server, make sure that the procedure name and start parameters are correct.

30015 SOMDERROR_BadTypeCode

Explanation: An invalid type code is being encountered while processing the request.

Programmer Response: Use a valid type codes.

30016 SOMDERROR_BadDescriptor

Explanation: The DSOM run time is encountering bad or missing information in the Interface Repository.

Programmer Response: Follow these steps:

1. Run the IR emitter to update the Interface Repository with the specified identifier.
2. Use the **irdump** utility to verify that the Interface Repository has been correctly updated.

30017 SOMDERROR_InvalidBaseProxyClass

Explanation: An invalid class is specified for a user-defined proxy.

Programmer Response: Modify the user-defined proxy so that it is derived from **SOMDClientProxy**.

30018 SOMDERROR_CouldNotStartThread

Explanation: The DSOM run time cannot create a thread.

Programmer Response: Bring down the DSOM application and free system resources.

30019 SOMDERROR_NoMessages

Explanation: SOMOA::execute_next_request, **Request::get_response**, or **Request::get_next_response** is calling, indicating not to wait but no requests or responses are pending. This might not be an error.

Programmer Response: Ignore the exception or change the method invocation to request a block until the message is available.

30020 SOMDERROR_UndeclaredException

Explanation: The application is raising an exception that is not specified in the method definition.

Programmer Response: Modify the method IDL to contain a raises expression for the exception.

30021 SOMDERROR_Marshaling Error

Explanation: The data passed to the DSOM marshaler is not valid or did not match the expected data type.

Programmer Response: Follow these steps:

1. Ensure that the data passed to the DSOM marshaler (for example, the parameters passed to a remote method invocation or the results returned from a remote invocation) are correctly initialized.

2. Ensure that all data structures are constructed according to the data type definitions in IDL for the method being invoked.

30022 SOMDERROR_ServerInterrupt

Explanation: The **SOMOA::interrupt_server** method is being invoked on the **SOMOA** object.

Programmer Response: Do not invoke **interrupt_server** on the **SOMOA** object.

30023 SOMDERROR_CommTimeOut

Explanation: Client and server processes are not communicating properly, which can happen if the **somdd** processes are not started on all participating server machines prior to starting the application processes.

Programmer Response: Follow these steps:

1. Verify the network setup. See *TCP/IP for MVS: Customization and Administration Guide* for additional information on network configurations.
2. Increase the values of **SOMDRECVWAIT** and **SOMDSENDWAIT**.

30024 SOMDERROR_CannotConnect

Explanation: There is no protocol over which the client can connect to the server.

Programmer Response: Ensure that the **SOMDPROTOCOLS** setting of the client has at least one entry in common with the **SOMDPROTOCOLS** setting under which the server was registered. Use the **regimpl** or **pregimpl** command to view the HOSTNAME setting for the server.

30025 SOMDERROR_BadConnection

Explanation: A previous connection with a server is being terminated because of a communications error. The request might not be successful.

Programmer Response: Verify the network setup. See *TCP/IP for MVS: Customization and Administration Guide* for more information about network configurations.

| **30026 SOMDERROR_UnauthSOMDD**

| **Explanation:** You tried to start the DSOM daemon (SOMDD), but are not authorized to
| start it.

| **Programmer Response:** The DSOM daemon is started as part of the SOM subsystem.
| Ensure that the SOM subsystem is started. If not, contact your system programmer.

30034 SOMDERROR_BadObjref

Explanation: The function or method is calling on an invalid object reference.

Programmer Response: Use or generate another reference to the target object.

30043 SOMDERROR_NoSOMDInit

Explanation: The application is attempting to create or access remote objects before DSOM initializes.

Programmer Response: Modify the application to call function **SOMD_Init** before making any run-time calls.

30044 SOMDERROR_CommunicationsError

Explanation: A communications error is occurring, which can happen if the **somdd** processes are not started on all participating server machines prior to starting the application processes.

Programmer Response: Follow these steps:

1. Verify the network setup. See SOMobject Developer Toolkit: Installation Guide for additional information on network setup.

30045 SOMDERROR_ImplRepIO

Explanation: An error is occurring while accessing Implementation Repository files. The files might be corrupted. This might indicate that the Implementation Repository files cannot be found or cannot be accessed.

Programmer Response: Follow these steps:

1. Verify that the SOMDDIR environment variable is set to a directory that grants read and write permissions to the DSOM user. (It is best if the directory name is fully qualified.)
2. If the SOMDDIR environment variable is not set, verify that the default directory, %SOMBASE%\etc\dsom, on OS/2 is set with the correct permissions.
3. Ensure that the files contained in the directory all have read and write permissions granted to the DSOM user.
4. If the preceding steps did not work, restore a backup version of the files, if available. If a backup version of the files is not available, rebuild the Implementation Repository.

30046 SOMDERROR_EntryNotFound

Explanation: The requested entry in the Implementation Repository cannot be found.

Programmer Response: Use the **regimpl** or **pregimpl** command to examine the contents. Reissue the command with the correct implementation ID, alias, or class.

30047 SOMDERROR_ClassNotFound

Explanation: One of the following applies:

- DSOM run time is failing to load the specified class, which can occur if the class name specified in calls to **somdCreate** or **find_any** is not associated with any server that is registered.
- The class libraries (DLLs) used to build the proxy class are statically linked to the program, but the DLL's **SOMInitModule** function is not properly initializing the class object, or has no **SOMInitModule**.
- A process cannot load the DLL associated with a particular class.

Programmer Response: Follow these steps:

1. Ensure that the class name is associated with at least one of the server implementations.
2. Ensure that the DLL resides in the directory specified in **LIBPATH** or **PATH** and that the class has an entry in the Interface Repository.
3. Verify that the DLL contains the **SOMInitModule** initialization function.
4. Ensure that the IDL for the class contains the **dllname** modifier, that this IDL has been compiled into the Interface Repository, and that the DLL name given by the **dllname** modifier can be loaded. (Use the **irdump** utility to determine whether a particular class appears in the IR.)

30048 SOMDERROR_ServerNotFound

Explanation: The input server alias cannot be found, or the **SOMDObjectMgr** object is raising an exception in response to a **somdFindServer**, **somdFindServerByName**, **somdFindServerByClass**, or **somdFindAnyServerByClass** invocation to indicate that the requested server cannot be found.

Programmer Response: Follow these steps:

1. Use the **regimpl** or **pregimpl** command to ensure that the requested server has been registered with the appropriate classes (in the case of **somdFindServerByClass** or **somdFindAnyServerByClass**).
2. Ensure that the application receiving the error can successfully contact the Naming Server into which this information is registered.

30049 SOMDERROR_ServerAlreadyExists

Explanation: A server process that is running has already registered itself with the DSOM daemon (**somdd**) using the implementation ID of the desired server program.

Programmer Response: If another instance of the server is not actually running, restart the DSOM daemon. (This exception can occur if a user-written server program terminates without notifying the DSOM daemon via a call to **deactivate_impl**).

30061 SOMDERROR_CtxNoPropFound

Explanation: The property being passed to **get_values** or **delete_values** is not in the **Context** object.

Programmer Response: Modify the application to pass a valid property name.

30066 SOMDERROR_BadParm

Explanation: An invalid parameter value is being passed to a function or method.

Programmer Response: See SOMobjects Developer Toolkit Programmer' s Reference for a description of the parameters to the specified function or method.

30070 SOMDERROR_AuthnFail

Explanation: DSOM cannot initialize the security run time in a client or server.

Programmer Response: If you do not want to use any secure DSOM servers, disable authentication by setting DISABLE_AUTHN=TRUE in the [somsec] stanza of the **somenv.ini** file and restart your application. If you want to communicate with secure DSOM servers, ensure that the Naming Server is running correctly and that you have successfully configured SOMobjects on your machine. Restart your application.

30072 SOMDERROR_SecurityFail

Explanation: The security initialization is failing.

Programmer Response: Ensure that the Security Server is running correctly and that you are logged on to LAN Server.

30080 SOMDERROR_DuplicateEntry

Explanation: DSOM cannot update the Implementation Repository. If you are attempting to add a new implementation definition, an alias already exists. If you are attempting to add a new class to an existing entry, a class is already associated with entry.

Programmer Response: Reissue the command with a new alias or class name.

30081 SOMDERROR_Internal

Explanation: An internal DSOM error is occurring.

Programmer Response: Retry the scenario. If the problem persists, gather information about the problem and follow your local procedures for resolving problems.

30082 SOMDERROR_BadUnionTag

Explanation: The union discriminant value does not match any of the defined cases and no default case is defined.

Programmer Response: See “Union Type” in *OS/390 V2R4.0 SOMobjects Programmer's Guide*.

30083 SOMDERROR_BadSequence

Explanation: An invalid sequence is being found while marshalling.

Programmer Response: See the “Template Types (Sequences and Strings)” in *OS/390 V2R4.0 SOMobjects Programmer's Guide* of the sequence IDL type. (It is an error for _length to be greater than _maximum. For bound sequences, it is an error to set _length or _maximum to be larger than the specified bound.)

30084 SOMDERROR_NotStreamable

Explanation: DSOM run time cannot marshal object as **pass_by_copy**. Processing will continue. An object reference is being sent instead.

Programmer Response: Verify that the **pass_by_copy** object is derived from **CosStream::Streamble** or that it is local (not a proxy).

30085 SOMDERROR_BadForeign

Explanation: A static foreign type is being produced through run-time conversion. Static foreign types should be used only in IDL.

Programmer Response: Do not use an **any** type that contains a static foreign type.

30086 SOMDERROR_NotForeignMarshaler

Explanation: An invalid class is specified for the dynamic foreign type.

Programmer Response: Derive the class from **SOMDDataMarshaler::ForeignMarshaler**.

30088 SOMDERROR_NamingNotActive

Explanation: The Naming Service is not active. Entries can be added to the Implementation Repository without updating the Naming Service. However, entries with information in both the Implementation Repository and the Naming Service must be kept consistent. Such entries cannot be updated or deleted if the Naming Service is not active.

Programmer Response: Verify that **GOSCFG** has been run by checking for **INSTALL** and **SOMNMOBJREF** entries in the [somnm] stanza of the configuration file. See the *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* for more information.

30089 SOMDERROR_WrongRefType

Explanation: The function or method is being invoked on an incompatible object reference. For example, **SOMOA::get_id** cannot be invoked on an object reference that is NULL, generated by the **create_SOM_ref** method, or a proxy.

Programmer Response: Modify the application to pass a compatible object reference.

30090 SOMDERROR_AbstractClass

Explanation: The method is being invoked on an abstract class.

Programmer Response: Modify the application code to invoke the method on a subclass.

30109 SOMDERROR_SOMDDNotRunning

Explanation: The DSOM daemon is not started.

Programmer Response: Start **somdd** and rerun the application.

35xxx-39xxx SOMDERROR_SOCKET

Explanation: A socket error occurred. To calculate the socket error code, subtract 3500 from the DSOM error code.

Programmer Response: Consult the technical reference for the implementor of the socket protocol in use. For OpenEdition MVS socket calls, socket error codes are defined in **errno.h**.

Life Cycle Service Error Codes

You might encounter the following error codes from the Life Cycle Service while an application is running.

52000 somLifeCycle_CREATE_SEMAPHORE_FAILURE

Explanation: A failure is occurring while creating a semaphore.

Programmer Response: Follow these steps:

1. Ensure that your system is not exceeding the maximum number of semaphores it is allowed to create.
2. Ensure that your system has threading support.

52001 somLifeCycle_REQUEST_SEMAPHORE_FAILURE

Explanation: A failure is occurring while requesting a semaphore, which can be caused by a time-out or because the semaphore was never created.

Programmer Response: Follow these steps:

1. Check the global environment after you create each object to ensure that it creates and initializes properly.
2. Ensure that your system has threading support.
3. Check for other threads in the process that might have acquired the semaphore and not released it.

52002 somLifeCycle_RELEASE_SEMAPHORE_FAILURE

Explanation: A failure is occurring while releasing a semaphore. A previous request to obtain the semaphore was unsuccessful, which also causes the release to be unsuccessful.

Programmer Response: Review the error log for any previous errors indicating a failure when requesting the semaphore.

52003 somLifeCycle_INVALID_LOCATION

Explanation: The parameter being passed is not a valid **somLifeCycle::Location** object. The object must be a **somLifeCycle::Location** or a descendent of the **somLifeCycle::Location** class.

Programmer Response: Invoke the **somIsA** method on the object to verify that the object reference is valid and supports the **somLifeCycle::Location** class.

52004 somLifeCycle_INVALID_CONSTRAINT_BUILDER

Explanation: The parameter being passed is not a valid **somLifeCycle::ConstraintBuilder** object. The object must be a **somLifeCycle::ConstraintBuilder** or a descendent of that class.

Programmer Response: Invoke the **somIsA** method on the object to verify that the object reference is valid and supports the **somLifeCycle::ConstraintBuilder** class.

52005 somLifeCycle_INVALID_STRING

Explanation: The parameter specified is not a valid **CosNaming::Istring**. The parameter cannot be NULL or an empty string.

Programmer Response: Modify the parameter to meet the **CosNaming::Istring** definition and submit the request again.

52006 somLifeCycle_CONSTRAINT_BUILDER_ALREADY _ASSOCIATED

Explanation: A **somLifeCycle::ConstraintBuilder** with the specified *key_kind* is already associated with the **somLifeCycle::FactoryFinder**.

Programmer Response: Follow these steps:

1. Invoke **list_constraint_builders** on the **somLifeCycle::FactoryFinder** to view the current list of **somLifeCycle::ConstraintBuilders** associated with the **somLifeCycle::FactoryFinder**.
2. Change the parameter specified to be a **somLifeCycle::ConstraintBuilder** with a *key_kind* that is not in the list returned by **list_constraint_builders**.

52007 somLifeCycle_CONSTRAINT_BUILDER_NOT _ASSOCIATED

Explanation: A **somLifeCycle::ConstraintBuilder** with the specified *key_kind* is not associated with the **somLifeCycle::FactoryFinde**r.

Programmer Response: Follow these steps:

1. Invoke **list_constraint_builders** on the **somLifeCycle::FactoryFinder** to view the current list of **somLifeCycle::ConstraintBuilders** associated with the **somLifeCycle::FactoryFinder**.
2. Change the parameter specified to match one of the **somLifeCycle::ConstraintBuilders** with a *key_kind* that is in the list returned by **list_constraint_builders**.

52008 somLifeCycle_INVALID_FACTORY_FILTER

Explanation: The parameter being passed is not a valid **somLifeCycle::FactoryFilter** object. The object must be a **somLifeCycle::FactoryFilter** or a descendent of the **somLifeCycle::FactoryFilter** class.

Programmer Response: Invoke the **somIsA** method on the object to verify that the object reference is valid and supports the **somLifeCycle::FactoryFilter** class.

52009 somLifeCycle_FACTORY_FILTER_ALREADY _ASSOCIATED

Explanation: A **somLifeCycle::FactoryFilter** with the specified *key_kind* is already associated with the **somLifeCycle::FactoryFinder**.

Programmer Response: Follow these steps:

1. Invoke **list_factory_filters** on the **somLifeCycle::FactoryFinder** to view the current list of **somLifeCycle::FactoryFilters** associated with the **somLifeCycle::FactoryFinder**.
2. Change the parameter specified to be a **somLifeCycle::FactoryFilter** with a *key_kind* that is not in the list returned by **list_factory_filters**.

52010 somLifeCycle_FACTORY_FILTER_NOT_ASSOCIATED

Explanation: A **somLifeCycle::FactoryFilter** with the specified *key_kind* is not associated with the **somLifeCycle::FactoryFinde**r.

Programmer Response: Follow these steps:

1. Invoke **list_factory_filters** on the **somLifeCycle::FactoryFinder** to view the current list of **somLifeCycle::FactoryFilters** associated with the **somLifeCycle::FactoryFinder**.
2. Change the parameter specified to match one of the **somLifeCycle::FactoryFilters**with a *key_kind* that is in the list returned by **list_factory_filters**.

52011 somLifeCycle_INVALID_FACTORY_FINDER

Explanation: The parameter being passed is not a valid **somLifeCycle::FactoryFinder** object. The object must be a **somLifeCycle::FactoryFinder** or a descendent of the **somLifeCycle::FactoryFinder** class.

Programmer Response: Invoke the **somIsA** method on the object to verify that the object reference is valid and supports the **somLifeCycle::FactoryFinder** class.

52012 somLifeCycle_INVALID_CRITERIA_SEQUENCE

Explanation: The **CosLifeCycle::Criteria** parameter specified is not a valid sequence.

Programmer Response: Modify the sequence to meet the following requirements:

- Ensure that the _length field is less than or equal to the _maximum field.
- If the _length field is greater than zero, the buffer must not be NULL.

52013 somLifeCycle_INVALID_SERVERID

Explanation: The parameter specified is not a valid **somLifeCycle::ServerId**.

Programmer Response: Modify the **ServerId** to meet the following requirements:

- The **ServerId** cannot be a NULL reference.
- The **ServerId** cannot be an empty string.

52014 somLifeCycle_INVALID_SOMOBJECT

Explanation: The parameter being passed is not a valid **SOMObject**.

Programmer Response: Invoke the **somIsA** method on the object to verify that the object reference is valid and supports the **SOMObject** class.

52015 somLifeCycle_INVALID_SERVERIDS_SEQUENCE

Explanation: The **somLifeCycle::ServerIds** parameter specified is not a valid sequence. The object must be a **SOMObject** or a descendent of the **SOMObject**.

Programmer Response: Modify the sequence to meet the following requirements:

- Ensure that the _length field is less than or equal to the _maximum field.
- If the _length field is greater than zero, the buffer must not be NULL.

52016 somLifeCycle_INVALID_KEY_SEQUENCE

Explanation: The **CosLifeCycle::Key** parameter specified is not a valid sequence.

Programmer Response: Modify the sequence to meet the following requirements:

- Ensure that the _length field is less than or equal to the _maximum field.
- If the _length field is greater than zero, the buffer must not be NULL.

52017 somLifeCycle_INVALID_OBJECT_INTERFACE_ID

Explanation: Anid string associated with the object class portion of the **CosLifeCycle::Key** is not valid. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string specified is object class, which is correct.

Programmer Response: Modify the id string to meet the following requirements:

- The string cannot be a NULL reference.
- The string cannot be an empty string.

52018 somLifeCycle_OBJECT_INTERFACE_ALREADY_SPECIFIED

Explanation: An object class is already specified in the **CosLifeCycle::Key**. Only one object class can be specified in the **CosLifeCycle::Key**.

Programmer Response: Follow these steps:

1. Review the **CosLifeCycle::Key** specified and remove the object class kind and the corresponding id value that is not needed.
2. Decrease the _length field of the sequence by one so that the number of items listed is correct.

52019 somLifeCycle_INVALID_CONSTRAINT_ID

Explanation: Anid string associated with the constraint portion of the **CosLifeCycle::Key** is not valid. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string specified is constraint, which is correct.

Programmer Response: Modify the id string to meet the following requirements:

- The string cannot be a NULL reference.
- The string cannot be an empty string.

52020 somLifeCycle_CONSTRAINT_ALREADY_SPECIFIED

Explanation: A constraint is already specified in the **CosLifeCycle::Key**. Only one constraint can be specified in the **CosLifeCycle::Key**.

Programmer Response: Follow these steps:

1. Review the **CosLifeCycle::Key** specified and remove the constraint kind and the corresponding id value that is not needed.
2. Decrease the _length field of the sequence by one so that the number of items listed is correct.

52021 somLifeCycle_INVALID_FACTORY_INTERFACE_ID

Explanation: Anid string associated with the factory class portion of the **CosLifeCycle::Key** is not valid. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string specified is factory class, which is correct.

Programmer Response: Modify the id string to meet the following requirements:

- The string cannot be a NULL reference.
- The string cannot be an empty string.

52022 somLifeCycle_FACTORY_INTERFACE_ALREADY _SPECIFIED

Explanation: A factory class is already specified in the **CosLifeCycle::Key**. Only one factory class can be specified in the **CosLifeCycle::Key**.

Programmer Response: Follow these steps:

1. Review the **CosLifeCycle::Key** specified class and remove the factory class kind and the corresponding id value that is not needed.
2. Decrease the _length field of the sequence by one so that the number of items listed is correct.

52023 somLifeCycle_INVALID_CB_ID

Explanation: An id string associated with the cb:*name* portion of the **CosLifeCycle::Key** is not valid. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string specified is cb:*name*, which is correct.

Programmer Response: Modify the id string so that it does not contain a NULL reference.

52024 somLifeCycle_INVALID_CONSTRAINT_BUILDER _NAME

Explanation: The name of a **somLifeCycle::ConstraintBuilder** specified in the **CosLifeCycle::Key** is not valid. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string that is required when specifying a **somLifeCycle::ConstraintBuilder** in the **CosLifeCycle::Key** is defined as cb:*name*, where *nam*e identifies the **somLifeCycle::ConstraintBuilder**. The id contains the value used with operations on the referenced **somLifeCycle::ConstraintBuilder**.

Programmer Response: Modify *name* to meet the following requirements:

- The string cannot be a NULL reference.
- The string cannot be an empty string.

52025 somLifeCycle_CONSTRAINT_BUILDER_NOT _FOUND

Explanation: The cb:*name* specified in the **CosLifeCycle::Key** is not the name of a **somLifeCycle::ConstraintBuilder** associated with the **somLifeCycle::FactoryFinder**. Only names of **somLifeCycle::ConstraintBuilder** objects associated with the **somLifeCycle::FactoryFinder** can be listed in the **CosLifeCycle::Key**.

Programmer Response: Follow these steps:

1. Invoke the **list_constraint_builders** method on the **somLifeCycle::FactoryFinder** to retrieve a list of all associated **somLifeCycle::ConstraintBuilder** objects.
2. Modify the **CosLifeCycle::Key** cb:*name* with one of the associated names.
3. Verify that the name selected is not already listed in another cb:*name* portion of the **CosLifeCycle::Key**, as duplicate names will result in the DUPLICATE_CONSTRAINT_BUILDER exception.

52026 somLifeCycle_DUPLICATE_CONSTRAINT_BUILDER

Explanation: The cb:*name* specified in the **CosLifeCycle::Key** is a duplicate of a cb:*name* previously specified. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string that is required when specifying a s**omLifeCycle::ConstraintBuilder** in the **CosLifeCycle::Key** is defined as cb:*name*, where *name* identifies the **somLifeCycle::ConstraintBuilder**. The id contains the value used with operations on the referenced **somLifeCycle::ConstraintBuilder**. Each associated **somLifeCycle::ConstraintBuilder** can be referenced only once in the **CosLifeCycle::Key**.

Programmer Response: Follow these steps:

1. Review the **CosLifeCycle::Key** and remove the duplicate cb:*name* kind and the corresponding id that is not needed.
2. Decrease the _length field of the sequence by one so that the number of items listed is correct.

52027 somLifeCycle_INVALID_FF_ID

Explanation: Anid string associated with the ff:*name* portion of the **CosLifeCycle::Key** is not valid. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string.The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string specified is ff:*nam*e, which is correct.

Programmer Response: Modify the id string so that the string is not a NULL reference.

52028 somLifeCycle_INVALID_FACTORY_FILTER_NAME

Explanation: The name of a **somLifeCycle::FactoryFilter** specified in the **CosLifeCycle::Key** is not valid. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string that is required when specifying a **somLifeCycle::FactoryFilter** in the **CosLifeCycle::Key** is defined as ff:*name*, where *name* identifies the **somLifeCycle::FactoryFilter**. The id contains the value used with operations on the referenced **somLifeCycle::FactoryFilter**.

Programmer Response: Modify *name* to meet the following requirements:

- The string cannot be a NULL reference.
- The string cannot be an empty string.

52029 somLifeCycle_FACTORY_FILTER_NOT_FOUND

Explanation: The ff:*name* specified in the **CosLifeCycle::Key** is not the name of a **somLifeCycle::FactoryFilter** that is associated with the **somLifeCycle::FactoryFinder**. Only names of **somLifeCycle::FactoryFilter** objects that are associated with the **somLifeCycle::FactoryFinder** can be listed in the **CosLifeCycle::Key**.

Programmer Response: Follow these steps:

1. Invoke the **list_factory_filters** method on the **somLifeCycle::FactoryFinder** to retrieve a list of all associated **somLifeCycle::FactoryFilter** objects.
2. Modify the **CosLifeCycle::Key** ff:*name* with one of the associated names.
3. Verify that the name selected is not already listed in another ff:*name* portion of the **CosLifeCycle::Key**, as duplicate names will result in the DUPLICATE_FACTORY_FILTER exception.

52030 somLifeCycle_DUPLICATE_FACTORY_FILTER

Explanation: The ff:*name* specified in the **CosLifeCycle::Key** is a duplicate of an ff:*name* previously specified. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind string that is required when specifying a **somLifeCycle::FactoryFilter** in the **CosLifeCycle::Key** is defined as ff:*name*, where *name* identifies the **somLifeCycle::FactoryFilter**. The id contains the value used with operations on the referenced **somLifeCycle::FactoryFilter**. Each associated **somLifeCycle::FactoryFilter** can be referenced only once in the **CosLifeCycle::Key**.

Programmer Response: Follow these steps:

1. Review the **CosLifeCycle::Key** and remove the duplicate ff:*name* kind and the corresponding id that is not needed.
2. Decrease the _length field of the sequence by one so that the number of items listed is correct.

52031 somLifeCycle_OBJECT_INTERFACE_MISSING

Explanation: The object class kind and corresponding id string are missing from the **CosLifeCycle::Key**. The **CosLifeCycle::Key** is a sequence consisting of **CosNaming::Name**, which includes a kind string and an id string. The kind values that can be specified in the **CosLifeCycle::Key** include object class, factory class, constraint, cb:*name*, and ff:*name*. The object class kind and the corresponding id value are required in the **CosLifeCycle::Key**. The other kind values are optional.

Programmer Response: Follow these steps:

1. Modify the **CosLifeCycle::Key** to include an object class kind and a corresponding id value.
2. Increase the _length field of the **CosLifeCycle::Key** so that the number of items in the key matches the _length field.

52032 somLifeCycle_METHOD_NOT_IMPLEMENTED

Explanation: No implementation is provided for the method invoked. In most cases, this exception results from a call to a **CosLifeCycle** method. Because the **CosLifeCycle** classes are abstract, the methods have not been implemented. The **somLifeCycle** classes are subclasses of the **CosLifeCycle** classes and provide implementations for the methods.

Programmer Response: Follow one of these steps:

- Create an instance of the **somLifeCycle** subclass and invoke the desired method on the **somLifeCycle** object rather than on the **CosLifeCycle** object.
- Subclass the class defining the desired method, override the method, and provide an implementation.

52033 somLifeCycle_OBJECT_NOT_COPYABLE

Explanation: The referenced object cannot be copied. In SOMobjects Version 3.0 of the Life Cycle Object Service, no default implementation is provided for the **copy** method.

Programmer Response: Subclass the **somLifeCycle::LifeCycleObject** class, override the **copy** method, and provide an implementation.

52034 somLifeCycle_OBJECT_NOT_MOVABLE

Explanation: The referenced object cannot be moved. In SOMobjects Version 3.0 of the Life Cycle Object Service, no default implementation is provided for the **move** method.

Programmer Response: Subclass the **somLifeCycle::LifeCycleObject** class, override the **move** method, and provide an implementation.

52035 somLifeCycle_NO_FACTORY_FOUND

Explanation: No **CosLifeCycle::Factory** objects can be found. This exception is raised in the following cases:

1. No **CosLifeCycle::Factory** objects are found that meet the requirements specified in the **CosLifeCycle::Key**.
2. No **somLifeCycle::FactoryFinder** object is associated with the **somLifeCycle::GenericFactory** and the **create_object** or **supports** method is being invoked.
3. The **create_object** method is being invoked on a **somLifeCycle::GenericFactory** with an associated **somLifeCycle::FactoryFinder**. **CosLifeCycle::Factory** objects are found that meet the requirements of the **CosLifeCycle::Key**, but none of them are able to create an instance of the desired object. The provided default implementation will attempt to create an instance of the desired object using the **CosLifeCycle::Factory** objects found that meet the **CosLifeCycle::Key** requirements. If the **CosLifeCycle::Factory** is a SOMClass or another **somLifeCycle::GenericFactory**, the implementation handles the creation for you.

Programmer Response: Each of the following responses correspond to a like number in the preceding Explanation section:

1. Verify that the **CosLifeCycle Factory** objects that are able to create instances of the desired object class specified in the **CosLifeCycle::Key** have been registered with the Naming Service. See the "Naming Service" chapter in *OS/390 V2R4.0 SOMobjects Object Services*.
2. **2.Invoke the set_factory_finder** method on the **somLifeCycle::GenericFactory** to associate a **somLifeCycle::FactoryFinder** object with it. The **somLifeCycle::FactoryFinder** object will then be used for the **create_object** and **supports** methods on the **somLifeCycle::GenericFactory**.
3. If the **CosLifeCycle::Factory** is a user-defined factory, subclass the **somLifeCycle::GenericFactoryAbstraction** class and override the **real_object_creation** method to perform the creation process on the user-defined factory. See the "Life Cycle Service" chapter in *OS/390 V2R4.0 SOMobjects Object Services* on how to perform this process.

52036 somLifeCycle_INVALID_CRITERIA_SPECIFIED

Explanation: The **CosLifeCycle::Criteria** parameter specified contains a value that is not valid. A **CosLifeCycle::Criteria** is a sequence of **CosLifeCycle::NameValuePair** structures. Each **CosLifeCycle::NameValuePair** consists of a **CosNaming::Istring***name* and an **any** type called *value*. The **any** type resolves to a _type field, which is a pointer to an instance of a **TypeCode** that represents the type of the value. The _value field is a pointer to the actual value, which is consistent with the _type field. The exception was raised because the _value field specified does not match the _type field of the **any***value*.

Programmer Response: Modify the *value* portion of the **CosLifeCycle::NameValuePair** so that the _value field is of the _type field specified. See the description of **any** type in *OS/390 V2R4.0 SOMobjects Programmer's Guide*.

Persistent Object Service Error Codes

You might encounter the following error codes from the Persistent Object Service while an application is running.

53000 somPersistenceMinor_GENERAL

Explanation: The method is raising a standard exception.

Programmer Response: Refer to the name of the standard exception to determine the cause of the problem. Gather information about the problem and follow your local procedures for resolving problems.

53001 somPersistenceMinor_SEMAPHORE_CREATE

Explanation: An error is occurring while creating a semaphore.

Programmer Response: Follow these steps:

1. Ensure that your system is not exceeding the maximum number of semaphores it is allowed to create.
2. Ensure that your system has threading support.

53002 somPersistenceMinor_SEMAPHORE_REQUEST

Explanation: An error is occurring while requesting a semaphore.

Programmer Response: Follow these steps:

1. Check the global environment after your create each object to ensure that it creates and initializes properly.
2. Ensure that your system has threading support.
3. Check for other threads in the process that might have acquired the semaphore and not released it.

53003 somPersistenceMinor_SEMAPHORE_RELEASE

Explanation: An error is occurring while releasing a semaphore.

Programmer Response: Review the error log for any previous errors indicating a failure when requesting a semaphore.

53004 somPersistenceMinor_DATA_NOT_FOUND

Explanation: The persistent object data cannot be found in the datastore referenced by the PID. This is a generic error that can be returned by any of the IBM-supplied PDS when the **restore** method cannot find the persistent data in the underlying datastore.

Programmer Response: You must refer to the data associated with the exception in the **somerror.log** file for datastore specific information about the error. Correct the error and retry the persistence method with a valid PID.

53008 somPersistenceMinor_POM_NOT_FOUND

Explanation: An error is occurring while creating a POM for this process.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

53009 somPersistenceMinor_DSTYPE_NOT_FOUND

Explanation: None of the entries in the POM data file matches the datastore type specified in the PID object.

Programmer Response: Check the datastore type in the PID. Check the **POS_POMDATA** environment variable. Check the POM data file.

53010 somPersistenceMinor_PROTOCOL_NOT_FOUND

Explanation: The object being passed to the **somPersistencePOM::POM** does not inherit from any of the protocol classes specified in the POM data file.

Programmer Response: Check the type of the object passed to the POM. Check the **POS_POMDATA** environment variable. Check the POM data file.

53011 somPersistenceMinor_POM_NIL_OBJECT

Explanation: The object being passed to the **somPersistence::POM** is NIL.

Programmer Response: Correct the parameter.

53012 somPersistenceMinor_POM_BAD_OBJECT

Explanation: The object being passed to the **somPersistence::POM** is not a valid **SOMObject**.

Programmer Response: Correct the object parameter.

53013 somPersistenceMinor_POM_NIL_PID

Explanation: The PID being passed to the **somPersistence::POM** is NIL.

Programmer Response: Correct the parameter.

53014 somPersistenceMinor_POM_BAD_PID

Explanation: The PID being passed to the **somPersistence::POM** is not a valid **SOMObject**.

Programmer Response: Correct the object parameter.

53015 somPersistenceMinor_POMDATA_SOMBASE_NOT_SET

Explanation: The **SOMBASE** environment variable is not set or is set incorrectly. It is required to determine the default location of the POM data file.

Programmer Response: Ensure the environment variable is set and that the first two characters are “//.”

53016 somPersistenceMinor_POMDATA_CANNOT_OPEN_FILE

Explanation: The POM data file cannot be opened for read access.

Programmer Response: Ensure that the POM data file exists in the location specified by the POS_POMDATA environment variable. Ensure that the file contains data and is readable.

53017 somPersistenceMinor_POMDATA_LINE_TOO_LONG

Explanation: One or more of the lines in the POM data file is too long.

Programmer Response: Edit the POM data file. Check for long lines, long comments, and trailing blanks.

53018 somPersistenceMinor_POMDATA_DSTYPE_MISSING

Explanation: One or more of the lines in the POM data file is missing the datastore type. The datastore type should be the second token found on the line.

Programmer Response: Correct the error in the POM data file.

53019 somPersistenceMinor_POMDATA_PDSTYPE_MISSING

Explanation: One or more of the lines in the POM data file is missing the PDS type. The PDS type should be the third token found on the line.

Programmer Response: Correct the error in the POM data file.

53020 somPersistenceMinor_POMDATA_TOO_MANY_ITEMS_PER_LINE

Explanation: One or more of the lines in the POM data file has too many tokens. Each line can have no more than three tokens.

Programmer Response: Correct the error in the POM data file.

53021 somPersistenceMinor_POMDATA_FILE_READ_ERROR

Explanation: An error is occurring while reading the POM data file.

Programmer Response: Ensure that the file exists and the contents are correct.

53022 somPersistenceMinor_POM_CANNOT_CREATE_PDS

Explanation: The **somPersistence::POM** object is attempting to create a PDS and failed.

Programmer Response: If you have modified the PDS_factory_finder attribute, ensure that it's set properly. The lookupPDS method of the somPersistencePOM class uses it as the target object when invoking the find_factory method of the somLifeCycle_FactoryFinder class. If you have not modified the PDS_factory_finder attribute, ensure that the PDS class is registered in the implementation repository.

53023 somPersistenceMinor_SOMOS_SERVER_OBJECT_NOT_FOUND

Explanation: The **SOMD_ServerObject** is not a **somOS::Server**. The **somPersistencePO::PO** objects must be created in a **somOS::Server** process.

Programmer Response: Either use **regimpl** to change the program and class of the server to **somOS::Server**, or create the PO object in another somOS::Server process.

53024 somPersistenceMinor_REINIT_METADATA_NOT_FOUND

Explanation: During reactivation, the persistence metadata cannot be found.

Programmer Response: Ensure that the object is passivated before it is removed from memory.

53025 somPersistenceMinor_REINIT_UNABLE_TO_CREATE_PID

Explanation: During reactivation, the PID cannot be created.

Programmer Response: Ensure that the PID class object is created.

53026 somPersistenceMinor_CAPTURE_METADATA_TOO_FULL

Explanation: During capture, the PID cannot be stored in the metadata.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

53027 somPersistenceMinor_POMDATA_EMPTY_OR_NOT_FOUND

Explanation: The POM data file was found unreadable or empty.

Programmer Response: Ensure that the POM data file exists in the location specified by the **POS_POMDATA** environment variable. Ensure that the file contains data and is readable.

53031 somPersistenceMinor_NULL_PIDSTRING

Explanation: The PID string you are attempting to set is NULL.

Programmer Response: Specify a valid PID string on the **set_PIDString** method.

53032 somPersistenceMinor_METHOD_IS_ABSTRACT

Explanation: The method is not implemented.

Programmer Response: Do not instantiate objects on abstract classes.

53100 somPersistencePOSIX_DEFAULT_MINOR_CODE

Explanation: The POSIX PDS is raising a standard exception.

Programmer Response: The standard exception indicates the nature of the problem. Gather information about the problem and follow your local procedures for resolving problems.

53101 somPersistencePOSIX_SEMAPHORE_CREATE

Explanation: The POSIX PDS is failing while creating a semaphore.

Programmer Response: Follow these steps:

1. Ensure that your system is not exceeding the maximum number of semaphores it is allowed to create.
2. Ensure that your system has threading support.

53102 somPersistencePOSIX_SEMAPHORE_REQUEST

Explanation: The POSIX PDS is failing while requesting a semaphore, which can be caused by a time-out or because the semaphore was never created.

Programmer Response: Follow these steps:

1. Check the global environment after you create each POSIX object to ensure that it creates and initializes properly.
2. Ensure that your system has threading support.
3. Check for other threads in the process that might have acquired the semaphore and not released it.

53103 somPersistencePOSIX_SEMAPHORE_RELEASE

Explanation: The POSIX PDS is failing while releasing a semaphore, which can be caused by a time-out or because the semaphore was never created.

Explanation: Review the error log for any previous errors indicating a failure when requesting a semaphore.

53104 somPersistencePOSIX_SEMAPHORE_DESTROY

Explanation: The POSIX PDS is failing while destroying a semaphore, which can be caused by a time-out or because the semaphore was never created.

Explanation: Review the error log for any previous errors indicating a failure when requesting a semaphore.

53105 somPersistencePOSIX_FILE_OPEN_ERROR

Explanation: The POSIX PDS is failing while opening or creating a file.

Programmer Response: Follow these steps:

1. Read the **somerror.log** file to find the error number (errno) returned by the file system. The error numbers are located after the minor code in the log. (This variable is operating system dependent.)
2. Read the **errno.h** file on the system where the error occurred to find the reason that corresponds to the error number.

53106 somPersistencePOSIX_FILE_CLOSE_ERROR

Explanation: The POSIX PDS is failing while closing a file.

Programmer Response: Follow these steps:

1. Read the **somerror.log** file to find the error number (errno) returned by the file system. The error numbers are located after the minor code in the log. (This variable is operating system dependent.)
2. Read the **errno.h** file on the system where the error occurred to find the reason that corresponds to the error number.

53107 somPersistencePOSIX_FILE_DELETE_ERROR

Explanation: The POSIX PDS is failing while deleting a file.

Programmer Response: Follow these steps:

1. Read the **somerror.log** file to find the error number (errno) returned by the file system. The error numbers are located after the minor code in the log. (This variable is operating system dependent.)
2. Read the **errno.h** file on the system where the error occurred to find the reason that corresponds to the error number.

53108 somPersistencePOSIX_FILE_IO_ERROR

Explanation: The POSIX PDS is failing while reading or writing to a file.

Programmer Response: Follow these steps:

1. Read the **somerror.log** file to find the error number (errno) returned by the file system. The error numbers are located after the minor code in the log. (This variable is operating system dependent.)
2. Read the **errno.h** file on the system where the error occurred to find the reason that corresponds to the error number.

53110 somPersistencePOSIX_INVALID_PATHNAME

Explanation: The PID_POSIX does not have a valid path name.

Programmer Response: Ensure that you call **_set_pathname** on the PID, setting it to a valid path or file name format.

53111 somPersistencePOSIX_FACTORY_NOT_VALID

Explanation: The factory being passed to the **initialize** method is not a valid **somExternalization::StreamFactory**.

Programmer Response: Ensure that you are passing a **somExternalization::StreamFactory** on the initialize call, or if you are setting the **streamFactory** attribute directly on the PDS, ensure that it is, or inherits from, a **somExternalization::StreamFactory**.

53112 somPersistencePOSIX_OBJ_NOT_STREAMABLE

Explanation: The object being passed to the POSIX PDS does not inherit from **CosStream::Streamable** as required.

Programmer Response: Ensure that the object (typically one of your PO objects) inherits from **CosStream::Streamable**.

53113 somPersistencePOSIX_NON_VALID_PID_CLASS

Explanation: The parameter being passed is not a valid POSIX PID.

Programmer Response: Ensure that the parameter passed to the PID is a **somPersistencePOSIX::PID_POSIX**.

53114 somPersistencePOSIX_PID_PATHNAME_NOT_SET

Explanation: The PID being passed to the POSIX PDS does not have its path name set.

Programmer Response: Set the path name on the PID before using it.

53200 somPersistenceBTREE_DEFAULT_MINOR_CODE

Explanation: The BTREE PDS is raising a standard exception. The standard exception indicates the nature of the problem.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

53201 somPersistenceBTREE_SEMAPHORE_CREATE

Explanation: The BTREE PDS is failing while creating a semaphore.

Programmer Response: Follow these steps:

1. Ensure that your system is not exceeding the maximum number of semaphores it is allowed to create.
2. Ensure that your system has threading support.

53202 somPersistenceBTREE_SEMAPHORE_REQUEST

Explanation: The BTREE PDS is failing while requesting a semaphore, which can be caused by a time-out or because the semaphore was never created.

Programmer Response: Follow these steps:

1. Check the global environment after you create each BTREE object to ensure that it creates and initializes properly.
2. Ensure that your system has threading support.
3. Check for other threads in the process that might have acquired the semaphore and not released it.

53203 somPersistenceBTREE_SEMAPHORE_RELEASE

Explanation: The BTREE PDS is failing while releasing a semaphore, which can be caused by a time-out or because the semaphore was never created.

Programmer Response: Review the error log for any previous errors indicating a failure when requesting a semaphore.

53204 somPersistenceBTREE_SEMAPHORE_DESTROY

Explanation: The BTREE PDS is failing while destroying a semaphore, which can be caused by a time-out or because the semaphore was never created.

Explanation: Review the error log for any previous errors indicating a failure when requesting a semaphore:

53205 somPersistenceBTREE_DATABASE_CREATE_ERROR

Explanation: The BTREE PDS is failing while creating the VSAM dataset. If the dataset does not exist, the BTREE PDS will try to create it before accessing it.

Programmer Response: Follow these steps:

1. Verify the dataset name is correct.
2. Verify that you have the proper authority to create the specified dataset.
3. Verify that you have enough space to allocate the specified dataset.
4. Read the **somerror.log** file to find the error code that VSAM returned to aid with the problem determination. The return codes are located after the minor code in the log.

53206 somPersistenceBTREE_DATABASE_OPEN_ERROR

Explanation: The BTREE PDS is failing while opening the VSAM dataset.

Programmer Response: Read the **somerror.log** file to find the error code that VSAM returned to aid with the problem determination. The return codes are located after the minor code in the log.

53207 somPersistenceBTREE_DATABASE_CLOSE_ERROR

Explanation: The BTREE PDS is failing while closing the VSAM dataset.

Programmer Response: Read the **somerror.log** file to find the error code that VSAM returned to aid with the problem determination. The return codes are located after the minor code in the log.

53208 somPersistenceBTREE_DATABASE_DELETE_KEY_NOT_FOUND

Explanation: The BTREE PDS is failing while deleting the persistent state of an object because it cannot find the key in the VSAM dataset.

Programmer Response:

1. Ensure that your key is valid and that you have not previously deleted the entry associated with that key.
2. Ensure that you are deleting the key from the same VSAM dataset in which you stored the key.
3. Read the **somerror.log** file to find the error code that VSAM returned to aid with the problem determination. The return codes are located after the minor code in the log.

53209 somPersistenceBTREE_DATABASE_DELETE_ERROR

Explanation: The BTREE PDS is failing while deleting the persistent state of an object. The key was found, but another problem exists.

Programmer Response: Read the **somerror.log** file to find the error code that VSAM returned to aid with the problem determination. The return codes are located after the minor code in the log.

53210 somPersistenceBTREE_DATABASE_READ_ERROR

Explanation: The BTREE PDS is failing while reading from the VSAM dataset.

Programmer Response: Follow these steps:

1. Ensure that you **call _set_datastore_name** on the PID, setting it to a valid VSAM dataset name.
2. Read the **somerror.log** file to find the error code that VSAM returned to aid with the problem determination. The return codes are located after the minor code in the log.

53211 somPersistenceBTREE_DATABASE_WRITE_ERROR

Explanation: The BTREE PDS is failing while writing to a VSAM dataset.

Programmer Response: Follow these steps:

1. Ensure that you **call _set_datastore_name** on the PID, setting it to a valid VSAM dataset name.
2. Read the **somerror.log** file to find the error code that VSAM returned to aid with the problem determination. The return codes are located after the minor code in the log.

53213 somPersistenceBTREE_INVALID_KEY_SIZE

Explanation: The size of the key is not valid.

Programmer Response: Correct the key size. Read the **sompbt.idl** file for the current maximum key size.

53215 somPersistenceBTREE_INVALID_DATASTORE_NAME

Explanation: The **PID_BTREE** does not have a valid VSAM dataset name.

Programmer Response: Ensure that you **call _set_datastore_name** on the PID, setting it to a valid VSAM dataset name.

53216 somPersistenceBTREE_INVALID_KEY

Explanation: The **PID_BTREE** does not have a valid key set.

Programmer Response: Ensure that you **call _set_object_key**, setting it to a valid key format.

53217 somPersistenceBTREE_FACTORY_NOT_VALID

Explanation: The factory being passed with the **initialize** method is not a valid **somExternalization::StreamFactory**.

Programmer Response: Ensure that you are passing a **somExternalization::StreamFactory** on the initialize call, or if you are setting the **streamFactory** attribute directly on the PDS, ensure that it is, or inherits from, a **somExternalization::StreamFactory**.

53218 somPersistenceBTREE_OBJ_NOT_STREAMABLE

Explanation: The object being passed to the BTREE PDS does not inherit from **CosStream::Streamable** as required.

Programmer Response: Ensure that the object (typically one of your PO objects) inherits from **CosStream::Streamable**.

53219 somPersistenceBTREE_NON_VALID_PID_CLASS

Explanation: The parameter being passed on the call is not a valid BTREE PID.

Programmer Response: Ensure that the parameter passed in for the PID is a **somPersistenceBTREE::PID_BTREE**.

53220 somPersistenceBTREE_PID_DATASTORE_NAME_NOT_SET

Explanation: The PID passed does not have its **datastore_name** set.

Programmer Response: Set the **datastore_name** on the PID before using it.

53221 somPersistenceBTREE_PID_KEY_NOT_SET

Explanation: The PID being passed does not have its **object_key** set.

Programmer Response: Set the **object_key** on the PID to a valid key before using it.

Externalization Service Error Codes

You might encounter the following error codes from the Externalization Service while an application is running.

54000 somStream_GENERAL

Explanation: The method is raising a standard exception.

Programmer Response: Refer to the name of the standard exception to determine the cause of the problem. Gather information about the problem and follow your local procedures for resolving problems.

54001 somStream_SEMAPHORE_CREATE

Explanation: An error is occurring while creating a semaphore.

Programmer Response: Follow these steps:

1. Ensure that your system is not exceeding the maximum number of semaphores it is allowed to create.
2. Ensure that your system has threading support.

54002 somStream_SEMAPHORE_REQUEST

Explanation: An error is occurring while requesting a semaphore.

Programmer Response: Follows these steps:

1. Check the global environment after you create each object to ensure that it creates and initializes properly.
2. Ensure that your system has threading support
3. Check for other threads in the process that might have acquired the semaphore and not released it.

54003 somStream_SEMAPHORE_RELEASE

Explanation: An error is occurring while releasing a semaphore.

Programmer Response: Review the error log for any previous errors indicating a failure when requesting a semaphore.

54004 somStream_FACTORY_FINDER_LOAD

Explanation: The SOM Life Cycle Service DLL cannot be loaded.

Programmer Response: Ensure that the **sommvs.sgosload(somlc)** file is accessible to the process.

54005 somStream_FACTORY_FINDER_NEW

Explanation: An error is occurring while creating a new factory finder.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

54006 somStream_FACTORY_FINDER_CLASS

Explanation: An error is occurring while retrieving the class of the specified factory finder.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

54007 somStream_FACTORY_FINDER_METHOD

Explanation: Cannot locate the **find_factory** method in the class.

Programmer Response: Ensure that the specified factory finder object supports a method called **find_factory**.

54008 somStream_FIND_FACTORY

Explanation: The **find_factory** method is failing.

Programmer Response: Ensure that the **find_factory** method returns a valid factory object.

54009 somStream_NEW_NOT_STREAMABLE

Explanation: The new object does not support the **CosStream::Streamable** class.

Programmer Response: Ensure that the object interface being specified in the stream supports the **CosStream::Streamable** class.

54010 somStream_OBJ_NOT_STREAMABLE

Explanation: The object does not support the **CosStream::Streamable** class.

Programmer Response: Ensure that the object being passed to the **externalize_ref** or **externalize_ref** method supports the **CosStream::Streamable** class.

54011 somStream_FACTORY_FINDER

Explanation: The factory finder parameter is not correct.

Programmer Response: Ensure that the factory finder parameter being passed to the method supports the **somLifeCycle::FactoryFinder** class.

54012 somStream_METHOD_IS_ABSTRACT

Explanation: The method is not implemented.

Programmer Response: Do not instantiate objects on abstract classes.

54013 somStream_OBJ_REPEAT_REF_MISMATCH

Explanation: The parameter to **internalize_existing** does not match the repeated reference found in the stream.

Programmer Response: Ensure that the topology of the graph of objects that is read from the stream is identical to the topology of the graph of objects that is written to the stream.

54014 somStream_UNABLE_TO_CREATE_STREAMIO

Explanation: An error is occurring while creating a **somStream::StandardStreamIO** object for use by the **Stream** object. The stream is unable to create a default **StreamIO** object.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

54015 somStream_STREAM_CREATE_WITH_TYPES

Explanation: An error is occurring while creating a stream and **StreamIO** with the **StreamFactory**. The **StreamFactory** is not able to create a stream.

Programmer Response: Gather information about the problem and follow your local

Programmer Response: procedures for resolving problems.

54016 somStream_UNABLE_TO_CREATE_STREAM

Explanation: An error is occurring while creating a **somExternalization::Stream** object for use by the **StreamIO** object. The **StreamIO** cannot create a default stream object.

Programmer Response: Gather information about the problem and follow your local

Programmer Response: procedures for resolving problems.

54017 somStream_INVALID_KEY_LENGTH

Explanation: The parameter being passed to the **write_key** method or the **write_object_tag** method is not valid.

Programmer Response: Ensure that the key passed to the **write_key** or **write_object_tag** method is correct. Check the **external_form_id** of the **Streamable** object.

54018 somStream_ALREADY_STREAMED_PARMS

Explanation: Parameters being passed to **already_streamed** are not valid.

Programmer Response: Ensure that you pass to the **already_streamed** method:

- a pointer to an object, and
- a pointer to a class object that is a parent of the object.

54019 somStream_BAD_BUFFER_PARAMETER

Explanation: The parameter to **set_buffer** has an invalid length or the header is incorrect.

Programmer Response: Ensure that the buffer you are attempting to set was produced by the **_get_buffer** method of the same type of **StreamIO**. Ensure that all of the fields of the octet sequence are set correctly.

54020 somStream_ICONV_FAILURE

Explanation: An error is occurring while converting the code page of the stream data.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

54021 somStream_READ_PASSED_END_OF_STREAM

Explanation: You are attempting to read past the end of the data in the **StreamIO**.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream. To read the stream twice, you must use the **reset** method.

54022 somStream_REPEAT_REF_NUMBER_NOT_FOUND

Explanation: A repeat reference is being read from the stream that has an invalid object number.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

54023 somStream_OBJ_NIL_DATA_MISMATCH

Explanation: A NIL data tag has been found in the stream, but a non-NIL object parameter is being passed.

Programmer Response: If you call **read_objec**t and the stream contains a NIL object indicator, ensure that you pass a NULL for the object parameter.

54024 somStream_OBJ_STRINGIFIED_REF_MISMATCH

Explanation: A stringified reference was found in the stream, but a non-NIL object parameter is being passed.

Programmer Response: If you call **read_object** and the stream contains a stringified reference, ensure that you pass a NULL for the object parameter or that the object you pass is identical to the object referred to by the stringified reference.

54025 somStream_FOUND_UNKNOWN_TAG

Explanation: An unknown object format tag is in the stream.

Programmer Response: Ensure that the stream contains an object. For example, if you **write_long** to the stream and then **read_object**, this error code is raised. If you override the **externalize_ref** method to write new tag values, you must also override the **internalizeInt** method to accept the new tag values.

54026 somStream_UNABLE_TO_READ_SHORT

Explanation: The **read_short** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54027 somStream_UNABLE_TO_READ_LONG

Explanation: The **read_long** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54028 somStream_UNABLE_TO_READ_USHORT

Explanation: The **read_unsigned_short** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54029 somStream_UNABLE_TO_READ_ULONG

Explanation: The **read_unsigned_long** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54030 somStream_UNABLE_TO_READ_STRING

Explanation: The **read_string** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54031 somStream_UNABLE_TO_READ_CHAR

Explanation: The **read_char** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54032 somStream_UNABLE_TO_READ_FLOAT

Explanation: The **read_float** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54033 somStream_UNABLE_TO_READ_DOUBLE

Explanation: The **read_double** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54034 somStream_UNABLE_TO_READ_OCTET

Explanation: The **read_octet** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54035 somStream_UNABLE_TO_READ_BOOLEAN

Explanation: The **read_boolean** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54036 somStream_UNABLE_TO_READ_OBJECT

Explanation: The **read_object** method is failing.

Programmer Response: Ensure that you are reading the same number and same type of items from the stream as were written to the stream.

54037 somStream_READ_FROM_EMPTY_STREAM

Explanation: You are trying to read data from a **StreamIO** that is empty.

Programmer Response: You must put data into the stream, by either writing to it or using the **set_buffer** method, before you attempt to read from it.

Naming Service Error Codes

You might encounter the following error codes from the Naming Service while an application is running.

55001 SOMNM_AbstractClass

Explanation: The specified class is defined as an abstract class. No implementation is provided for this class. You cannot invoke methods on an abstract class.

Programmer Response: Use the concrete class shipped with the product.

55002 SOMNM_RandomFailed

Explanation: The file name generator is failing to generate a unique file name to be used by the Naming Service, which can happen if the directory pointed to by the SOMDDIR environment variable contains a large number of files (that is, the Naming Service has approached its limits).

Programmer Response: Try deleting unnecessary naming contexts. If the problem persists, gather information about the problem and follow your local procedures for resolving problems.

55003 SOMNM_DestroyFailed

Explanation: An error is occurring while destroying a naming context.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

55004 SOMNM_ImplLimit

Explanation: The implementation is reaching its internal limits.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

55101 SOMNM_DatabaseError

Explanation: An error is occurring while accessing a naming database. The SOMDDIR environment variable in the **somenv.ini** file specifies the location of these databases.

Programmer Response: Ensure what the naming database VSAM files are located are valid.

55102 SOMNM_DataBaseOpenError

Explanation: An error is occurring while opening a naming database. The SOMDDIR environment variable in the **somenv.ini** file specifies the location of these databases

Programmer Response: Ensure that the naming database VSAM files are located are valid.

55103 SOMNM_TypecodeError

Explanation: An error is occurring while manipulating a property value.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

55104 SOMNM_DatabaseCreateError

Explanation: An error is occurring while creating a naming database. The SOMDDIR environment variable in the **somenv.ini** file specifies the location of these databases.

Programmer Response: Ensure that the drive, path, and directory where naming database files are located are valid. Ensure that the directory permissions allow files to be created.

55201 SOMNM_FilterInternalError

Explanation: An error is occurring in the Naming Service while processing a search request.

Programmer Response: Try restarting the Naming Server. If that does not work, gather information about the problem and follow your local procedures for resolving problems.

55202 SOMNM_ConstraintError

Explanation: An error is occurring in one of the **find** methods while processing the constraint

Programmer Response: Ensure that the constraint string specified in the search satisfies the grammar.

55203 SOMNM_ConstraintTooLong

Explanation: The constraint that is specified for the **find** method is exceeding internal limits.

Programmer Response: Invoke the **find** method using a smaller constraint. Try splitting the query into sub-queries.

55301 SOMNM_ANYInternalError

Explanation: An error is occurring while trying to use the **any** data type.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

55401 SOMNM_NoComponent

Explanation: The method is attempting to operate on a non-existent name component, which can happen if you try to access a name component that is not defined in the **LName** object.

Programmer Response: Ensure that the specified component was inserted using the **insert_component** method.

55402 SOMNM_OverFlow

Explanation: The method is attempting to insert a name component into a **LName** object and is exceeding the maximum number of components in an **LName** object. The maximum number of components allowed is 16.

Programmer Response: Reduce the number of name components in the **LName** object.

55404 SOMNM_NotSet

Explanation: The method is attempting to retrieve the id string or the kind string within a name component. The id string or the kind string might not be set, which might be acceptable.

Programmer Response: Do nothing if the calling client is using the Naming Service in such a way that the id string or kind string does not need to be set; otherwise, check your application logic and retry the operation.

55405 SOMNM_NotFound

Explanation: The Naming Service cannot resolve the name, which can happen if the name was never bound to the naming context or if the name was unbound from the context.

Programmer Response: If the name is a compound name, ensure that all the sub- contexts in the compound name are accessible from the **NamingContext** object on which the resolve operation was performed.

55407 SOMNM_AlreadyBound

Explanation: The **bind** method (or its variations) is attempting to associate an object to a name that was previously bound in the same context.

Programmer Response: Use the **rebind** method (or its variations) to replace an existing binding or use the **unbind** method followed by the **bind** method to achieve the same result.

55408 SOMNM_NotEmpty

Explanation: A **destroy** method is being invoked on a naming context that has bindings. All bindings in the naming context must to be removed before the naming context can be destroyed.

Programmer Response: Use the **unbind** method to unbind objects.

55412 SOMNM_PropertyNotFound

Explanation: A method call to get the property cannot find the specified property. If the method was a batch operation such as **get_properties**, then the last property in the list that cannot be found results in this exception.

Programmer Response: Do nothing if the calling client is using the Naming Service in such a way that the specified property does not need to be set; otherwise, check your application logic and retry the operation.

55415 SOMNM_IllegalConstraint

Explanation: The constraint specified for the **find** methods is not syntactically valid.

Programmer Response: Correct the syntax of the constraint.

55416 SOMNM_BindingNotFound

Explanation: One of the **find** methods cannot find a binding that satisfies the specified search constraint. This might be valid response that requires no further action.

Programmer Response: Ensure that the *distance* parameter that was specified for the search is valid and reasonable.

Security Service Error Codes

You might encounter the following error codes from the Security Service while an application is running.

56001 SOMSEC_ERRCODE_AuthnFail

Explanation: The principal making a remote method invocation is failing authentication.

Programmer Response: Follow these steps:

1. Ensure that you are logged on to LAN Services. Alternatively, you can log off LAN Server and run your application without authentication, bearing in mind that a secure server will reject unauthenticated method invocations.
2. Restart your application.

56006 SOMSEC_ERRCODE_NoRegistry

Explanation: The Security Server cannot create or open its database. Only one Security Server can be running at a time.

Programmer Response: Ensure that a Security Server is not already running. Also, ensure that you have set the SOMDDIR environment variable in your **somenv.ini** file to a directory in which you have write access.

56007 SOMSEC_ERRCODE_BadRegistry

Explanation: An error is occurring while opening the Registry database.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

56009 SOMSEC_ERRCODE_ObjectNotFound

Explanation: Initialization of Security Services is failing because needed resources cannot be obtained from the Naming Services.

Programmer Response: Ensure that **SOMobjects** is properly configured.

56010 SOMSEC_ERRCODE_AuthnRequired

Explanation: The application is making an unauthenticated remote method request to a secure server.

Programmer Response: Log on to LAN Services and restart your application. Alternatively, restart the application server in a non-secure mode using the **regimpl** or **pregimpl** command.

56011 SOMSEC_ERRCODE_InvalidSessId

Explanation: An error is occurring in **SOMobjects**.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

56013 SOMSEC_ERRCODE_CannotLoadDll

Explanation: A LAN Services dynamic link library (DLL) cannot be loaded and used by Security Services. The name of the DLL in question is logged in the message that accompanies this error. The Security Services cannot perform authentication without this DLL. Your application will continue running, unauthenticated.

Programmer Response: If your application requires authentication, as the case will be if it needs services from a secure server, you need to stop your application and reinstall LAN Services on your machine before restarting your application.

56014 SOMSEC_ERRCODE_FunctionMissing

Explanation: A LAN Services dynamic link library (DLL) is available but is not the appropriate DLL for authentication purposes. Your application will continue running, unauthenticated.

Programmer Response: Upgrade or reinstall LAN Services if you need to use the authentication facility provided by the Security Services.

56015 SOMSEC_ERRCODE_NoLoginContext

Explanation: You did not log on to LAN Services before starting your application. Your application will continue running, unauthenticated.

Programmer Response: If you need to use the authentication facility provided by the Security Services, stop the application, log on, and restart the application.

56017 SOMSEC_ERRCODE_UnknownError

Explanation: A subsystem used by the Security Service is giving an unrecognized return code.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

56018 SOMSEC_ERRCODE_IOError

Explanation: The Security Server is encountering an error while using its database. There can be only one Security Server running at a time.

Programmer Response: Check that a Security Server is not already running. Also, ensure that you have set the SOMDDIR environment variable in your **somenv.ini** file to a directory in which you have write access.

56019 SOMSEC_ERRCODE_NotSecure

Explanation: The Security Service is attempting to authenticate your method requests to a non-secure server.

Programmer Response: Do nothing if your application expects the server to be non-secure. Otherwise, stop the server and restart it in a secure mode. Restart your application.

56020 SOMSEC_ERRCODE_BadSequence

Explanation: An error is occurring in the Security Service.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

56021 SOMSEC_ERRCODE_UnrecognizedMsgNr

Explanation: An error is occurring in the Security Service.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

56022 SOMSEC_ERRCODE_UnrecognizedMsgFormat

Explanation: An error is occurring in the Security Service.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

Object Services (OS) Server Error Codes

You might encounter the following error codes from the Object Services Server while an application is running.

57000 SOMOS_Server_Already_Running

Explanation: The initialization of the persistence database is failing because the OS Server is already running. Only one instance of the OS Server associated with a particular implementation ID or implementation alias can be running in a system at any given time.

Programmer Response: Either start the OS Server using a different implementation alias or terminate the existing OS Server.

57001 SOMOS_Database_Directory_Not_Accessible

Explanation: The OS Server cannot access the directory specified by the SOMDDIR environment variable in the **somenv.ini** file.

Programmer Response: Verify that the path, drive, and directory are valid and accessible and restart the OS Server process again.

57002 SOMOS_Unable_To_Open_Master_Database

Explanation: The OS Server cannot open or create the master database **somosdb.dat**. This database contains the references to the metadata database and the attribute persistence database for each implementation of an OS Server. The SOMDDIR environment variable in the **somenv.ini** file specifies where the master database is located on your system.

Programmer Response: Ensure that the database drive, path, and directory are available and accessible.

57003 SOMOS_Unable_To_Open_Metadata_Database

Explanation: The OS Server cannot access the metadata database. This database contains the persistence object information for the OS Server. The SOMDDIR environment variable in the **somenv.ini** file specifies where the metadata database is located.

Programmer Response: Ensure that the database drive, path, and directory are available and accessible.

57004 SOMOS_Unable_To_Open_Attribute_Persist_Database

Explanation: The OS Server cannot access the attribute persistence database. The SOMDDIR environment variable in the **somenv.ini** file specifies where the attribute persistence database is located.

Programmer Response: Ensure that the database drive, path, and directory are available and accessible.

57005 SOMOS_Unable_To_Find_Attribute_Persist_Database

Explanation: The OS Server cannot initialize because the attribute persists database cannot be found. The SOMDDIR environment variable in the **somenv.ini** file specifies where the attribute persistence database is located.

Programmer Response: Ensure that the database drive, path, and directory are available and accessible.

57006 SOMOS_Unable_To_Find_Metadata_Database

Explanation: The OS Server cannot initialize because the metadata database cannot be found. The SOMDDIRenvironmentvariable in the **somenv.ini** file specifies where the metadata database is located.

Programmer Response: Ensure that the database drive, path, and directory are available and accessible.

57007 SOMOS_Usage_Error

Explanation: An invalid command is being entered to start the OS Server. The following parameters are valid: **somossvr** [-**i**] [-**d**] -**a** [**impl_alias** | **impl_uuid**], where -**i** is to initialize the OS Server the first time it is started, -**d** is for debugging purposes, and -**a** is to specify the implementation alias for the OS Server.

Programmer Response: Specify the correct parameters to start the OS Server.

57008 SOMOS_Unable_To_Find_Implementation_Definition

Explanation: The OS Server cannot find the implementation definition as it is specified to the OS Server.

Programmer Response: Correct the implementation definition as it is specified to the **somossvr** program. If you start the OS Server manually, ensure that the -**a** option specifies a valid implementation alias or that the implementation ID is valid, which is specified without any parameters.

57009 SOMOS_Impl_Is_Ready_Failed

Explanation: The **_impl_is_ready()** call is failing within the OS Server.

Programmer Response: See "DSOM Error Codes" on page 7-4 for this function for a further description of possible error conditions.

57010 SOMOS_SOMD_Init_Failed

Explanation: The **SOMD_Init()** function is failing within the OS Server.

Programmer Response: See "DSOM Error Codes" on page 7-4 for this function for a further description of possible error conditions.

57012 SOMOS_Server_Exit_Abnormal.

Explanation: The OS Server process is exiting abnormally, which can be caused by an unrecoverable error.

Programmer Response: Check previous error log messages for this server process for an explanation of the abnormal exit. Correct the problem and restart the OS Server.

57100 SOMOS_Class_Name_Error

Explanation: A method call to **make_persistent_ref** cannot find the class name specified.

Programmer Response: Pass a valid class name to this method and retry the operation.

57101 SOMOS_UUID_Create_Error

Explanation: The **make_persistent_ref** methodcannot create a UUID.

Programmer Response: See "SOM Kernel Error Codes" on page 7-2 for a further description on why the call **somCreateUUID()** failed and for possible solutions.

57102 SOMOS_Not_In_Cache

Explanation: An attempt to find an object in the internal cache is unsuccessful.

Programmer Response: Ensure that the item is in the persistence database, which makes use of the internal cache. The object reference might not be persistent, if that is the case, ensure that a persistence reference is created for the object with a call to **make_persistent_ref()**.

57103 SOMOS_Add_To_Cache_Error

Explanation: The **make_persistent_ref** methodisattempting to add the object to the internal cache but the object already exists in the cache. This method has already been performed on the object.

Programmer Response: Do not use the **make_persistent_ref** method on the specified object. Correct your program and retry the operation.

57104 SOMOS_Add_Object_To_Database_Error

Explanation: The **make_persistent_ref** method is attempting to add the object to the persistence database but is encountering an error. The SOMDDIR environment variable in the **somenv.ini** file specifies where the databases are located.

Programmer Response: Ensure that the drive, path, and directory containing the persistence database are valid and accessible.

57105 SOMOS_Database_Error

Explanation: The **delete_ref** method cannot delete the object reference from the persistence database. The environment variable SOMDDIR in the **somenv.ini** file specifies where the persistence database is located.

Programmer Response: Ensure that the persistence database is still accessible.

57106 SOMOS_Add_Item_To_Database_Error

Explanation: The method is encountering an error while saving metadata to the persistence database. The persistence database might not be available and accessible.

Programmer Response: Ensure that the drive, path, and directory containing the persistence databases are valid and accessible.

57107 SOMOS_Database_Synchronization_Error

Explanation: The method is encountering an error while synchronizing the persistence database. The persistence database might not be available and accessible.

Programmer Response: Ensure that the drive, path, and directory containing the persistence databases are valid and accessible.

57108 SOMOS_Get_Item_From_Database_Error.

Explanation: The method is encountering an error while retrieving an item from the persistence database. The persistence database might not be available and accessible.

Programmer Response: Ensure that the drive, path, and directory containing the persistence databases are valid and accessible.

57109 SOMOS_Bad_Reference_Data.

Explanation: The reference data is not from a **somOS::Server** class, which can happen if the **SOMObjFromRef** method is being used on the **somOS::Server**.

Programmer Response: Correct the reference data and try again.

57110 SOMOS_Internal_Failure

Explanation: The **somOS::Server** class is encountering an error. The OS Server is attempting to create a transaction object with the Transaction Management Service but is unsuccessful.

Programmer Response: Ensure that Transaction Management Service is installed and available.

57111 SOMOS_Get_Object_From_Database_Error

Explanation: The **somOS::Server** class cannot retrieve an object from the persistence database. This error can occur in the **SOMObjFromRef** method when the object reference is not found in the persistence database.

Programmer Response: Ensure that the object has a persistent reference by using the **make_persist_ref** method.

57112 SOMOS_Passivate_Object_Not_Found

Explanation: The **passivate_all_object** method is being called and not all objects can be found. Only one error message will be logged even though several objects might not be found during this method call.

Programmer Response: Ensure that the object reference is valid and that the specified object can be passivated.

57113 SOMOS_Internal_Cache_Error.

Explanation: The method call is encountering an internal cache problem while attempting to find an item in the cache.

Programmer Response: Use the **make_persistent_ref** method call to ensure that the object has a persistent reference.

57114 SOMOS_Database_Initializing_Error

Explanation: The **somOS::Server** class is encountering a problem while initializing the persistent database. This problem can occur if the persistence database directory is not valid and accessible.

Programmer Response: Ensure that the SOMDDIR environment variable in the **somenv.ini** file that specifies the location of the persistence databases is valid and accessible.

57115 SOMOS_Database_Open_Error

Explanation: The **somOS::Server** class is encountering a problem opening the internal persistence databases while initializing the class. This problem can occur if the persistence database directory is not valid or accessible.

Programmer Response: Ensure that the SOMDDIR environment variable in the **somenv.ini** file that specifies the location of the persistence databases is valid and accessible.

57116 SOMOS_Cache_Setup_Error

Explanation: The **somOS::Server** class is encountering an error during initialization while setting up the internal cache.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57117 SOMOS_Creating_Transaction_Object_Error

Explanation: The **somOS::Server** class cannot create a Transaction Management Service object.

Programmer Response: Ensure that Transaction Management Service is installed and active when you perform this operation.

57200 SOMOS_Locking_Failure

Explanation: The method is attempting to create a semaphore lock and is failing.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57201 SOMOS_Cache_Error

Explanation: The method is attempting to initialize the internal cache and is failing.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57202 SOMOS_Database_Error

Explanation: The method is attempting to create the persistence database and is failing.

Programmer Response: Follow these steps:

1. Ensure that the drive, path, and directory are valid and accessible where the persistence database is located.
2. Ensure that the SOMDDIR environment variable in the **somenv.ini** that specifies where the persistence databases are located is valid and accessible.

57203 SOMOS_Not_In_Database_Error

Explanation: The specified attribute cannot be found in the database.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57204 SOMOS_Not_In_IR_Error

Explanation: The attribute definition cannot be found in the Implementation Repository.

Programmer Response: Ensure that the Implementation Repository path specified by the **SOMIR** environment variable is still valid and accessible and that it contains the required Implementation Repositories for your environment.

57205 SOMOS_IR_Access_Error

Explanation: An error is occurring while accessing the Implementation Repository to retrieve an attribute type.

Programmer Response: Ensure that the Implementation Repository path specified by the **SOMIR** environment variable is still valid and accessible and that it contains the required Implementation Repositories for your environment.

57206 SOMOS_Decode_Error

Explanation: An error is occurring while decoding an attribute.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57207 SOMOS_Encode_Error

Explanation: An error is occurring while encoding an attribute.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57208 SOMOS_Parent_Error

Explanation: An error is occurring while calling the parent class of the object.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57209 SOMOS_Persist_Manager_Error

Explanation: An error is occurring while initializing the Persistence Manager class.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57210 SOMOS_No_Persist_Manager_Error

Explanation: An error is occurring while accessing the Persistence Manager class.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57211 SOMOS_OS_Server_Error

Explanation: An error is occurring while calling the **somOS::Server** class.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57212 SOMOS_Mutex_Initialization_Error

Explanation: An error is occurring while initializing a semaphore.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57213 SOMOS_Persistent_Reference_Error

Explanation: An error is occurring while creating or deleting a persistence reference.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57214 SOMOS_No_Database_Object_Error

Explanation: The Persistence Manager class cannot create a database object.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57215 SOMOS_No_Cache_Object_Error

Explanation: The Persistence Manager class cannot create a cache object.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57216 SOMOS_Persist_Manager_Already_Initialized_Error.

Explanation: An error is occurring while initializing the Persistence Manager because it is already initialized.

Programmer Response: Gather information about the problem and follow your local procedures for resolving problems.

57217 SOMOS_Persist_Manager_Create_Error

Explanation: The Persistence Manager is failing to create a new process.

Programmer Response: Ensure that you have enough resources available in your system.

Metaclass Framework Error Codes

You might encounter the following messages from the Metaclass Framework while an application is running.

60019

Explanation: An attempt is being made to construct a class with **SOMMSingleInstance** as a metaclass constraint, which can occur indirectly because of the construction of a derived metaclass. The initialization of the class is failing because **somInitMIClass** defined by **SOMMSingleInstance** is in conflict with another metaclass that is overriding **somNew** Method. That is, some other metaclass is already claiming the right to return the value for **somNew**.

60029

Explanation: An attempt is being made to construct a class with **SOMMSingleInstance** as a metaclass constraint. (This might occur indirectly because of the construction of a derived metaclass). The initialization of the class is failing because **somInitMIClass** defined by **SOMMSingleInstance** is in conflict with another metaclass that is overriding **somFree** method.. That is, some other metaclass is already claiming the right to override **somFree**.

60069

Explanation: A **SOMMBeforeAfter** metaclass must override both the **sommBeforeMethod** and **sommAfterMethod**. An attempt is being made to create a **SOMMBeforeAfter** metaclass where only one of the preceding methods is overridden.

60079

Explanation: An attempt is being made to subclass a non-subclassable metaclass. Only the SOM kernel is allowed to create subclasses of non-subclassable metaclasses when building a derived metaclass.

60089

Explanation: An attempt is being made to create a proxy class with more than two parents. A proxy class can have either two parents (a proxy class and the class of a target object) or one parent (a proxy class: in this case, a special proxy class is being created).

60099

Explanation: An attempt is being made to create a proxy class for which the first parent was not a descendant of **SOMMProxyForObject** class. The first parent of all proxy classes must be a descendant of **SOMMProxyForObject**.

Appendix A. Troubleshooting Hints

This chapter documents Diagnosis, Modification or Tuning Information.

Troubleshooting Hints is divided into five sections:

- “Interface Repository Dump (IRDUMP) Utility”
- “Checking the DSOM Setup” on page A-2
- “Analyzing Problem Symptoms” on page A-2
- “Unexplained Program Crashes” on page A-4
- “Reporting Problems to IBM” on page A-5

The following hints may prove helpful as you develop and test your SOM application.

Interface Repository Dump (IRDUMP) Utility

If you run applications that use the SOMobjects runtime library, you might find using the Interface Repository Dump (IRDUMP) utility useful.

The SOMobjects Interface Repository (IR) contains information about SOMobjects for MVS classes. Applications that use the runtime library might not run successfully if the IR that you are using is corrupted or back level. To help with diagnosis, you can use the IRDUMP utility to print the contents of the IR.

Figure A-1 shows an example of JCL to run the IRDUMP utility. This JCL can be found in SOMMVS.SGOSJCL(GOSIRDMP).

```
//IRDUMP   JOB <job card parameters>
//*
//*================================================================*
//*  Job to execute irdump utility in batch job.                   *
//*================================================================*
//*
// SET PARMS=''
//*SET PARMS='TCPIPSockets'
//*SET PARMS='dAnimal'
//*
// SET SOMENV=SOMMVS.SGOSPROF(GOSENV1)
// SET DSOMLIB=SOMMVS.SGOSLOAD
// SET LIBPRFX=CEE.V1R5M0
//*
//*================================================================*
//*
//IRDUMP  EXEC  PGM=IRDUMP,REGION=40M,TIME=NOLIMIT,
//  PARM='&PARMS'
//*
//STEPLIB   DD DSN=&DSOMLIB.,DISP=SHR
//          DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//*
//SOMENV    DD DSN=&SOMENV.,DISP=SHR
//*
//SYSPRINT  DD SYSOUT=*
//*
```

Figure A-1. JCL to dump the IR output

The //SOMENV DD statement must point to a data set that has the SOMIR global variable. The SOMIR global variable lists the current interface repository files. The sample global variable data set provided with SOMobjects is SOMMVS.SGOSPROF(GOSENV1). Your installation might have chosen to use a different data set.

Checking the DSOM Setup

This checklist will help you ensure that the DSOM environment is set up correctly.

- Ensure that your networking hardware and software are properly installed. For SOMobjects on OS/390, TCP/IP is supported. The hosts file must contain an entry for both the client and server machines.

  For TCP/IP, verify the networking setup for your particular networking product. For more information on your network's hardware/software setup, see Program Directory for Exclusive OS/390 Elements Version 1 Release 3.

- For all application class libraries to be loaded dynamically, IDL containing dllname modifiers must be compiled into the Interface Repository designated by **SOMIR**. (This includes any application-specific subclasses of **SOMDServer** and **SOMDClientProxy**.) You can verify that a class exists in the Interface Repository by executing **irdump** *className*. All application libraries must also be found in the MVS search order determined by your STEPLIB.
- An implementation must be registered with DSOM by running the **regimpl** utility. The server must be registered on the machine on which it will run. (A server cannot be registered from a different machine.) The **SOMDDIR**, **SOMDPROTOCOLS**, **HOSTNAME** and **SOMDPORT** settings in effect at the time the server was registered must be the same as those in effect when the server (and its associated SOM subsystem) are executed. Before a server can be registered the Naming Service and Security Service must be configured, using the **GOSCFG** JCL procedure. For more information on this, refer to the *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*. The server also needs to be registered to WLM and RACF.
- Verify that all class libraries and networking libraries can be found in the MVS search order determined by your STEPLIB. Both the client and server machines need a DLL for each class. For machines that will run only client programs (and not servers), a *stub* DLL may be used instead.
- Ensure that the contents of the **SOMDDIR** directory and the **SOMNMOBJREF** setting are consistent with one another (that is, they were all produced from the same run of the **GOSCFG** JCL procedure). See *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* for more information. Further, the **SOMDPROTOCOLS**, **HOSTNAME** and **SOMDPORT** settings of the name server and its associated SOM subsystem should match those in effect when the **GOSCFG** JCL procedure was run.

Analyzing Problem Symptoms

Check any DSOM error codes returned. If none of the error codes apply to your situation, check the following suggestions for possible solutions.

- **Problem:** A SOMDERROR_ClassNotFound error is returned by a server when creating a remote object.

  Solution: This error may result if the DLL for the class cannot be found. Verify that:

  – the interface of the object can be found in the IR
  – the class name is spelled correctly and is appropriately scoped (for example, the Printer class in the **PrintServer** module must have the identifier **PrintServer::Printer**)

  This error can also result if the DLL for the class does not provide a **SOMInitModule** that initializes all the classes it contains or if SOMInitModule has not been exported.

  This error may result when the class libraries used to build the proxy class are statically linked to the program, but the *className***NewClass** procedures have not been called to initialize the classes in the library's **SOMInitModule** function.

  A DLL built with the **imod** emitter provides a proper **SOMInitModule**.

- **Problem:** Following a method call, the SOM run-time error message, "A target object failed basic validity checks during method resolution" is displayed.

  Solution: Usually this means that the method call was invoked using a bad object pointer, or the object has been corrupted.

- **Problem:** A remote object has an attribute or instance variable that is, or contains, a pointer to a value in memory (for example, a string, a sequence, an any). The attribute or instance variable value is set by the client with one method call. When the attribute or instance variable is queried in a subsequent method call, the value referenced by the pointer is "garbage.."

  Solution: This may occur because DSOM makes a copy of argument values in a client call, for use in the remote call. The argument values are valid for the duration of that call. When the remote call is completed, the copies of the argument values are freed. Therefore, an object must allocate storage for, and copy, parameter values that are used to set attributes.

- **Problem:** A method defines a char * parameter that is used to pass a string input value to an object. The object attempts to print the string value, but it appears to be "garbage.."

  Solution: DSOM will support method arguments that are pointers (pointer types are a SOM extension), by de-referencing the pointer in the call, and copying the base value. The value and the related pointer are reconstructed on the server before the actual method call is made.

  While (char *) is commonly used to refer to NULL-terminated strings in C programs, (char *) could also be a pointer to a single character or to an array of characters. Thus, DSOM interprets the argument type literally as a pointer-to-one-character.

  To correctly pass strings or array arguments, the appropriate CORBA type should be used (for example, string or char foo[4]).

- **Problem:** A protection exception occurs when passing an any argument to a method call, where the any value is a string, array, or object reference. The NamedValues used in DII calls use any fields for the argument values.

  Solution: This error may occur because the _*value* field of the any structure does not contain the address of a pointer to the target string, array, or object reference, as it should. (A common mistake is to set the _*value* field to the address of the string, array, or object reference itself. To have DSOM transmit an any whose _*value* field is NULL, set the _*type* field of the any to tk_null.)

- **Problem:** When a server program or a server object makes a call to get_id or to get_SOM_object on a SOMDObject, a BAD_OPERATION exception is returned with an error code of SOMDERROR_WrongRefType.

  Solution: This error may occur when the operation **get_id** is called on a **SOMDObject** that does not have any user-supplied **ReferenceData** (that is, the **SOMDObject** is a proxy, is nil, or is a simple “SOM ref” created by **create_SOM_ref**). Likewise, this error may occur when the operation **get_SOM_object** is called on a **SOMDObject** that was not created by the **create_SOM_ref** method.

- **Problem:** A protection exception occurs when a SOMD_Uninit call is executed.

  Solution: This error could occur if the application has already freed any of the DSOM run-time objects that were allocated by the **SOMD_Init** call, such as **SOMD_ImplRepObject** or **SOMD_ORBObject**.

Unexplained Program Crashes

Verify that the DSOM environment is configured properly, as described in *OS/390 V2R4.0 SOMobjects Programmer's Guide* and in the *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*. Verify that all class libraries can be found in the MVS search order determined by your STEPLIB. Verify that the contents of the Interface Repository, specified by SOMIR, are correct (use the IRDUMP utility to examine the Interface Repository to determine that the contents are correct). Verify that the contents of the Implementation Repository, specified by SOMDDIR, are correct (use the REGIMPL utility to examine the Implementation Repository to determine that the contents are correct). Verify that the SOM subsystem is running on all participating server machines .

- **Problem:** Problems occur after changing the setting of SOMDPROTOCOLS, HOSTNAME, or SOMDPORT.

  Solution: After a change to SOMDPROTOCOLS, or to the HOSTNAME or SOMDPORT setting within a protocol stanza, the Naming Service must be reconfigured (using **SOMCFG** JCL), and all application servers must be re-registered using regimpl.

- **Problem:** The somossvr server program terminates immediately after starting.

  Solution: This will occur if the **somossvr** (for example, GOSOSSVR JCL procedure) server program has not been initialized the first time it is run. Unlike the generic **somdsvr** server program, the **somossvr** server program must be run manually at least once before it can be started on demand by the SOM subsystem. For this initial run, it must be given the **-i** command-line argument.

- **Problem:** A server program using a SOMobjects object services terminates.

  Solution: This will occur if the server JCL procedure was not registered to use both the **gosossvr** (or equivalent) and the **somOS::Server** server class. This is required by all the SOMobjects object services (Persistent Object Service, Naming Service, Security Service, and LifeCycle Service).

- **Problem:** A method can be invoked successfully on a server the first time, but subsequent invocations return garbage data to the caller.

  Solution: This occurs if the target object's implementation does not adhere to the caller-owned policy for parameter memory management, but the object's IDL does not contain the proper memory-management modifiers (for example,

object_owns_result, **object_owns_parameters**). As a result, the DSOM runtime in the server process is freeing memory that the target object still holds a pointer to.

- **Problem:** The server abends just after an application method (invoked remotely) returns.

  Solution: This can occur if the application method being invoked in the server does not properly initialize the inout/out parameters and the return value data structures according their declared IDL types. As part of marshalling results back to the caller, DSOM traverses all returned data structures according to their declared IDL types, copying the data therein to an interprocess message. If any values in those data structures are invalid (for example, if one contains an invalid pointer), this will cause DSOM to abend.

- **Problem:** A client application abends just after invoking a method remotely, and the remote method is never executed in the server.

  Solution: This can occur if the client does not properly initialize the in and inout parameters according to their declared IDL. As part of marshalling parameters to the server, DSOM traverses all in/inout parameters according to their declared IDL types, copying the data therein to an interprocess message. If any values in those data structures are invalid (for example, if one contains an invalid pointer), this will cause DSOM to abend.

- **Problem:** A client application abends while making a remote method call, just after the remote method has been executed in the server, but before control returns to the client application.

  Solution: This can occur if the client does not provide storage for out parameters and return values according to their declared IDL types. The client is responsible for allocating (but not necessarily initializing) the top-level storage for all out and return values. If this storage is not allocated, then DSOM may abend when attempting to store the out/return values from the remote call in the storage it thinks has been provided.

- **Problem:** A client application times out while waiting for communication back from a distributed request.

  Solution: This problem may be caused by insufficient resources defined for your TCP/IP network. Modify the **SOMAXCONN** configuration parameter in your **TCPIP.PROFILE** dataset. See *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* for more information on this problem and solution.

Reporting Problems to IBM

You should have the following available when reporting problems to an IBM service representative:

- Error log
- Trace data set(s). One for each active process involved in the method request (such as MVS client, SOM subsystem, naming server, security server, application server).
- Error messages from sysout (if any)
- LE dump from CEEDUMP (if any)
- Pertinent output from sysprint (if any).

Glossary

This glossary defines important terms and abbreviations used across the products that deliver Objects on MVS. This glossary does not contain general data processing terms that can be found in other published works. If you do not find the term you are looking for, refer to the Index or to the *Dictionary of Computing*, SC20-1699.

This glossary includes terms and definitions from:

- The *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.
- The American Production and Inventory Control Society, Inc., *APICS Dictionary*, Sixth Edition, 1987. Definitions are identified by the symbol (APICS) after the definition.

Note: In the following definitions, words shown in *italics* are terms for which separate glossary entries are also defined.

A

abstract class (SOM definition). A *class* that is not designed to be instantiated, but serves as a *base class* for the definition of subclasses. Regardless of whether an abstract class inherits *instance data* and *methods* from *parent classes*, it will always introduce methods that must be *overridden* in a *subclass,* in order to produce a class whose objects are semantically valid.

abstract class (C++ definition). (1) A class with at least one pure virtual function that is used as a base class for other classes. The abstract class represents a concept; classes derived from it represent implementations of the concept. You cannot have a direct object of an abstract class. (See also *base class.*) (2) A class that allows polymorphism. There can be no objects of an abstract class; they are only used to derive new classes.

access. Determines whether or not a class member is accessible in an expression or declaration.

accessory. A choice or option offered to customers for customizing the end product. An accessory is typically shipped with the end product, not built into it. Examples of accessories are product documentation, car floor mats, and camera cases. One accessory may be usable with more than one end product. (Product documentation may be usable for a product family.) Accessories may be *mandatory* (documentation) or *optional* (camera cases). When more than one choice is offered for an accessory, the choices are referred to as *variants*. For example, product documentation variants would typically provide national language support, variants for camera cases would provide multiple price or function options. See also *feature*.

address. A name, label, or number identifying a location in storage.

addressing mode (AMODE). In MVS, a program attribute that refers to the address length that a program is prepared to handle upon entry. *IBM.*

aggregate. (1) An array or a structure. (2) A compile-time option to show the layout of a structure or union in the listing. (3) An array or a class object with no private or protected members, no constructors, no base classes, and no virtual functions. (4) In programming languages, a structured collection of data items that form a data type. *ISO-JTC1.*

aggregate type. A user-defined data type that combines basic types (such as, char, short, float, and so on) into a more complex type (such as structs, arrays, strings, sequences, unions, or enums).

alignment. The storing of data in relation to certain machine-dependent boundaries.

American National Standards Institute (ANSI). An organization consisting of producers, consumers, and general interest groups that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. (A)

AMODE (addressing mode). In MVS, a program attribute that refers to the address length that a program is prepared to handle upon entry. *IBM.*

ancestor class. A *class* from which another class inherits *instance methods, attributes,*and *instance variables*, either directly or indirectly. A direct descendant of an ancestor class is called a *child class, derived class,* or *subclass.* A direct ancestor of a class is called a *parent class, base class*, or *superclass*.

angle brackets. The characters < (left angle bracket) and > (right angle bracket). When used in the phrase "enclosed in angle brackets", the symbol < immediately precedes the object to be enclosed, and > immediately follows it. When describing these characters in the portable character set, the names <less-than-sign> and <greater-than-sign> are used. *X/Open*.

argument. (1) A parameter passed between a calling program and a called program. *IBM*. (2) In a function call, an expression that represents a value that the calling function passes to the function specified in the call. Also called *parameter*. (3) In the shell, a parameter passed to a utility as the equivalent of a single string in the *argv* array created by one of the *exec* functions. An argument is one of the options, option-arguments, or operands following the command name. *X/Open*.

array. In programming languages, an aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting. *IBM*.

attribute. Generically, the data that defines a property or characteristic of an object. In IDL, a specialized syntax for declaring “set” and “get” methods. Method names corresponding to attributes always begin with “_set_” or “_get_”. An attribute name is declared in the body of the *interface statement* for a class. Method procedures for get/set methods are automatically defined by the *SOM Compiler* unless an attribute is declared as “noget/noset”. Likewise, a corresponding *instance variable* is automatically defined unless an attribute is declared as “nodata”. IDL also supports “readonly” attributes, which specify only a “get” method.

B

base class. See *parent class.*

base class (C++ definition). A class from which other classes are derived. May itself be derived from another base class. (See also *abstract class*.)

based on. The use of existing classes for implementing new classes.

batch. See *batch job*.

batch job. A job submitted as a predefined series of actions to be performed with little or no interaction between user and system.

behavior (of an object). The *methods* that an *object* responds to. These methods are those either introduced or inherited by the *class* of the object. See also *state.*

bindings. Language-specific macros and procedures that make implementing and using SOM classes more convenient. These bindings offer a convenient interface to SOM that is tailored to a particular programming language. The SOMobjects for MVS Compiler generates binding files for C and C ++. These binding files include an *implementation template* for the class and two header files, one to be included in the class's implementation file and the other in client programs.

block (in SACL). In *dialog tags*, an opening tag, its corresponding ending tag, and everything between them.

block (in C++). (1) In programming languages, a compound statement that coincides with the scope of at least one of the declarations contained within it. A block may also specify storage allocation or segment programs for other purposes. *ISO-JTC1*. (2) A string of data elements recorded or transmitted as a unit. The elements may be characters, words or physical records. *ISO Draft*. (3) The unit of data transmitted to and from a device. Each block contains one record, part of a record, or several records.

braces. The characters { (left brace) and } (right brace), also known as *curly braces*. When used in the phrase "enclosed in (curly) braces" the symbol { immediately precedes the object to be enclosed, and } immediately follows it. When describing these characters in the portable character set, the names <left-brace> and <right-brace> are used. *X/Open*.

brackets. The characters [(left bracket) and] (right bracket), also known as *square brackets*. When used in the phrase "enclosed in (square) brackets" the symbol [immediately precedes the object to be enclosed, and] immediately follows it. When describing these characters in the portable character set, the names <left-bracket> and <right-bracket> are used. *X/Open*.

built-in. (1) A function which the compiler will automatically inline instead of making the function call, unless the programmer specifies not to inline. (2) In programming languages, pertaining to a language object that is declared by the definition of the programming language; for example the built-in function SIN in PL/I, the predefined data type INTEGER in FORTRAN. *ISO-JTC1*. Synonymous with predefined. *IBM*.

C

C library. A system library that contains common C language subroutines for file access, string operators, character operations, memory allocation, and other functions.

C set+. An object-oriented application development package for building OS/2 and AIX solutions in C/MVS and C++/MVS.

case label. The word **case** followed by a constant expression and a colon. When the selector evaluates the value of the constant expression, the statements following the case label are processed.

character. (1) A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes. *ANSI*. (2) A sequence of one or more bytes representing a single graphic symbol or control code. This term corresponds to the ISO C standard term *multibyte character* (multi-byte character), where a single-byte character is a special case of the multi-byte character. Unlike the usage in the ISO C standard, *character* here has no necessary relationship with storage space, and *byte* is used when storage space is discussed. *X/Open*. *ISO.1*.

character set. (1) A finite set of different characters that is complete for a given purpose; for example, the character set in ISO Standard 646, "7-bit Coded Character Set for Information Processing Interchange". *ISO Draft*. (2) All the valid characters for a programming language or for a computer system. *IBM*. (3) A group of characters used for a specific reason; for example, the set of characters a printer can print. *IBM*.

child class (SOM definition). A class that inherits *instance methods, attributes,* and *instance variables* directly from another class, called the *parent class, base class*, or *superclass,* or indirectly from an *ancestor class.* A child class may also be called a *derived class* or *subclass.*

class (SOM definition). A way of categorizing *objects* based on their behavior (the *methods* they support) and shape (memory layout). A class is a definition of a generic object. In SOM, a class is also a special kind of object that can manufacture other objects that all have a common shape and exhibit similar behavior. The specification of what comprises the shape and behavior of a set of objects is referred to as the "definition" of a class. New classes are defined in terms of existing classes through a technique known as *inheritance*. See also *class object.*

class (C++ definition). (1) A C++ aggregate that may contain functions, types, and user-defined operators in addition to data. Classes may be defined hierarchically, allowing one class to be derived from another, and may restrict access to its members. (2) A user-defined data type that can contain both data representations (data members) and functions (member functions).

class library. A collection of C++ classes.

class manager. An *object* that acts as a run-time registry for all SOM *class objects* that exist within the current process and which assists in the dynamic loading and unloading of class libraries. A class implementor can define a customized class manager by subclassing *SOMClassMgr* class to replace the SOM-supplied SOMClassMgrObject. This is done to augment the functionality of the default class-management registry (for example, to coordinate the automatic quiescing and unloading of classes).

class method. A class method is a *method* that a *class object* responds to (as opposed to an *instance method*). A class method that class <X> responds to is provided by the *metaclass* of class <X>. Class methods are executed without requiring any *instances* of class <X> to exist, and are frequently used to create instances of the class.

class object. The run-time *object* representing a SOM *class.* In SOM, a class object can perform the same behavior common to all *objects*, inherited from *SOMObject*.

class variable. *Instance data* of a *class object*. All instance data of an *object* is defined (through either introduction or *inheritance*) by the object's class. Thus, class variables are defined by *metaclasses*.

client. (1) A user. (2) A functional unit that receives shared services from a server.

client code. (Or *client program* or *client*.) An application program, written in the programmer's preferred language, which invokes *methods* on *objects* that are *instances* of SOM *classes*.

CLIST. An executable list of TSO commands.

code page (SACL definition). A specification of code points for each graphic character in a set or in a collection of graphic character sets. Within a code page, a code point can have one and only one specific meaning.

collection. (1) An abstract class without any ordering, element properties, or key properties. All abstract classes are derived from collection. (2) In a general sense, an implementation of an abstract data type for storing elements.

column. (1) One of two or more vertical arrangements of lines, positioned side by side on a page or screen.

(T) (2) A vertical arrangement of characters or other expressions. (A) (3) A character position within a print line or on a display. The positions are numbered from 1, by 1, starting at the leftmost character position and extending to the rightmost position. (4) In SQL*, the vertical part of a table. A column has a name and a particular data type; for example, character, decimal, or integer.

comment. (1) Any descriptive information whose format depends on the context in which the comment exists (a source code comment, a folder comment, etc.). (2) In programming languages, a language construct for the inclusion of text in a program and having no impact on the execution of the program. Comments are used to explain certain aspects of the program. (I) (3) A statement used to document a program or file. Comments include information that may be helpful in running a job or reviewing an output listing. (4) Synonymous with computer program annotation, note, remark. Comments serve as documentation instead of as instructions. They are not processed by a compiler.

completion status. A code that indicates a review or deliverable has completed its cycle.

const. (1) An attribute of a data object that declares the object cannot be changed. (2) A keyword that allows you to define a variable whose value does not change.

constant. (1) In programming languages, a language object that takes only one specific value. *ISO-JTC1*. (2) A data item with a value that does not change. *IBM*.

constant expression. An expression having a value that can be determined during compilation and that does not change during program execution.

constructor (C++ definition). A class member function with the same name as its class, used to construct class objects and sometimes to initialize them.

context expression. An optional expression in a method's IDL declaration, specifying identifiers whose value (if any) can be used during SOM's *method resolution* process and/or by the *target object* as it executes the *method procedure*. If a context expression is specified, then a related Context parameter is required when the method is invoked. (This Context parameter is an *implicit parameter* in the IDL specification of the method, but it is an explicit parameter of the method's procedure.) No SOM-supplied methods require context parameters.

conversion (SACL definition). The transformation of data into a format that is acceptable to an application or system.

conversion (C++ definition). (1) In programming languages, the transformation between values that represent the same data item but belong to different data types. Information may be lost due to conversion since accuracy of data representation varies among different data types. *ISO-JTC1*. (2) The process of changing from one method of data processing to another or from one data processing system to another. *IBM*. (3) The process of changing from one form of representation to another; for example to change from decimal representation to binary representation. *IBM*. (4) A change in the type of a value. For example, when you add values having different data types, the compiler converts both values to a common form before adding the values.

CORBA. The Common Object Request Broker Architecture established by the Object Management Group. IBM's *Interface Definition Language* used to describe the *interface* for SOM classes is fully compliant with CORBA standards.

current working directory. (1) A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash. *X/Open*. *ISO.1*. (2) In DOS, the directory that is searched when a filename is entered with no indication of the directory that lists the filename. DOS assumes that the current directory is the root directory unless a path to another directory is specified. *IBM*. (3) In the OS/2 operating system, the first directory in which the operating system looks for programs and files and stores temporary files and output. *IBM*. (4) In the AIX operating system, a directory that is active and that can be displayed. Relative path name resolution begins in the current directory. *IBM*.

customize. To adapt or modify an application to fit a unique environment using build-time facilities provided as part of the application. Two kinds of customizing are available. The information services department customizes the application to fit the information processing system facilities available; for example, libraries, databases, workstations, communications facilities, and so on. The application specialists customize enterprise and application profiles to match the business policies and practices applicable to all users in the enterprise, or all users of one application. An example is changing a user interface dialog or panel. Users may personalize applications to further suit their unique needs. See also *personalize*.

D

data definition (DD). (1) In C/MVS and C++/MVS, a definition that describes a data object, reserves storage for a data object, and can provide an initial value for a data object. A data definition appears outside a function or at the beginning of a block statement. *IBM*. (2) A program statement that describes the features of, specifies relationships of, or establishes context of, data. *ANSI*. (3) A statement that is stored in the environment and that externally identifies a file and the attributes with which it should be opened.

data structure. The internal data representation of an implementation.

data type. The properties and internal representation that characterize data.

database. A collection of data with a given structure for accepting, storing, and providing, on demand, data for multiple users. (T)

debug. To detect, locate, and correct errors in a program.

declaration. (1) In C/MVS and C++/MVS, a description that makes an external object or function available to a function or a block statement. *IBM*. (2) Establishes the names and characteristics of data objects and functions used in a program.

declarator. Designates a data object or function declared. Initializations can be performed in a declarator.

declare. To identify the variable symbols to be used at preassembly time.

default. A value, attribute, or option that is assumed when no alternative is specified by the programmer.

default implementation. One of several possible implementation variants offered as the default for a specific abstract data type.

define directive. A preprocessor statement that directs the preprocessor to replace an identifier or macro invocation with special code.

definition. (1) A data description that reserves storage and may provide an initial value. (2) A declaration that allocates storage, and may initialize a data object or specify the body of a function.

delete. (1) The keyword **delete** identifies a free store deallocation operator. (2) The delete operator is used to destroy objects created by **new**.

derivation. In C++, to derive a class, called a derived class, from an existing class, called a base class.

derived class (SOM definition). See *subclass* and *subclassing.*

derived class (C++ definition). A class that inherits the proper base class become members of a derived class object. You can add additional data members and member functions to the derived class. A derived class object can be manipulated as if it is a base class object. The derived class can override virtual functions of the base class.

derived metaclass. (Or *SOM-derived metaclass*.) A *metaclass* that SOM creates automatically (often even when the *class* implementor specifies an explicit metaclass) as needed to ensure that, for any code that executes without *method-resolution* error on an *instance* of a given class, the code will similarly execute without method-resolution error on instances of any *subclass* of the given class. SOM's ability to derive such metaclasses is a fundamental necessity in order to ensure binary compatibility for client programs despite any subsequent changes in class *implementations*.

descendant. Any class that has the current class in its inheritance hierarchy.

descriptor (SACL definition). A word or phrase used to categorize or index information. Synonymous with keyword. (T)

descriptor (SOM definition). (Or *method descriptor*.) An ID representing the identifier of a *method* definition or an *attribute* definition in the Interface Repository. The IR definition contains information about the method's return type and the type of its arguments.

descriptor (C++ definition). PL/I control block that holds information such as string lengths, array subscript bounds, and area sizes, and is passed from one PL/I routine to another during run time.

dialog tag. One of a set of statements used to specify online panel format in the SACL system. Dialog tags are also used to specify help, messages, varclasses, and function key text. The dialog tags are processed by the dialog tag parser to produce a dialog tag extract (DTX) member. This member, in turn, is used as input to the user interface server (UIS) at run time.

dialog tag parser. An SACL program that processes dialog tags to produce output dialog tag extract (DTX) members. The dialog tag parser and dialog tags are used for panel customization.

difference. Given two sets A and B, the difference (A-B) is the set of all elements contained in A but not in B. For bags, there is an additional rule for duplicates: If bag P contains an element *m* times and bag Q contains the same element *n* times, then, if *m>n*, the difference contains that element *m-n* times. If *m≤n*, the difference contains that element zero times.

directive. A message (a pre-defined character constant) received by a *replica* from the Replication Framework. Indicates a potential failure situation.

dispatch-function resolution. Dispatch-function resolution is the slowest, but most flexible, of the three *method-resolution* techniques SOM offers. Dispatch functions permit method resolution to be based on arbitrary rules associated with an *object*'s *class*. Thus, a class implementor has complete freedom in determining how methods invoked on its *instances* are resolved. See also *dispatch method* and *dynamic dispatching.*

dispatch method. A *method* (such as somDispatch or somClassDispatch) that is invoked (and passed an argument list and the ID of another method) in order to determine the appropriate *method procedure* to execute. The use of dispatch methods facilitates *dispatch-function resolution* in SOM applications. See also *dynamic dispatching*.

DLL (dynamic link library). A collection of functions and variables accessed by external programs. A DLL identifies all functions, methods, and attributes of classes.

double-quote. The character ", also known as *quotation mark*. *X/Open*.

driver. A system or device that enables a functional unit to operate.

dump. To copy data in a readable format from main or auxiliary storage onto an external medium such as tape, diskette, or printer. *IBM*.

dynamic. Pertaining to an operation that occurs at the time it is needed rather than at a predetermined or fixed time. *IBM*.

dynamic binding (SACL definition). The automatic selection of routines based on the class of an object at execution time.

dynamic link library (DLL). A collection of functions and variables accessed by external programs. A DLL identifies all functions, methods, and attributes of classes.

E

element. The component of an array, subrange, enumeration, or set.

emitter. Generically, a program that takes the output from one system and converts the information into a different form. Using the Emitter Framework, selected output from the *SOM Compiler* (describing each syntactic unit in an *IDL source file*) is transformed and formatted according to a user-defined template. Example emitter output, besides the implementation template and language bindings, might include reference documentation, class browser descriptions, or “pretty” printouts.

empty string. (1) A string whose first byte is a null byte. Synonymous with null string. *X/Open*. (2) A character array whose first element is a null character. *ISO.1*.

enabler. A set of build-time tools provided to simplify and control the process of creating programs or data sets.

encapsulation (SOM definition). An object-oriented programming feature whereby the implementation details of a class are hidden from client programs, which are only required to know the *interface* of a *class* (the signatures of its *methods* and the names of its *attributes*) in order to use the class's methods and attributes.

encoder/decoder. In the Persistence Framework, a *class* that knows how to read/write the persistent object format of a *persistent object*. Every persistent object is associated with an Encoder/Decoder, and an encoder/decoder object is created for each *attribute* and *instance variable*. An Encoder/Decoder is supplied by the Persistence Framework by default, or an application can define its own.

enterprise. The entire business organization under discussion. It may consist of one or many establishments, divisions, plants, warehouses, and so on. With respect to systems, enterprise refers to a single installation.

entry class. In the Emitter Framework, a *class* that represents some syntactic unit of an *interface* definition in the *IDL source file.*

entity. Any concrete or abstract thing of interest, including associations among things; for example, a person, object, event, or process that is of interest in the context under consideration, and about which data may be stored in a database. (T)

enumeration type. A distinct data type that is not an integral type. An enumeration type defines a set of enumeration constants.

enumerator. An enumeration constant and its associated value.

Environment parameter. A *CORBA*-required parameter in all *method procedures*, it represents a memory location where exception information can be returned by the *object* of a method invocation. [Certain methods are exempt (when the class contains a modifier of callstyle=oidl), to maintain upward compatibility for client programs written using an earlier release.]

error. A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

event (SACL definition). An SACL application activity that is the result of a direct or an indirect user action.

exception (SACL definition). In programming languages, an abnormal situation that may arise during execution, that may cause a deviation from the normal execution sequence, and for which facilities exist in a programming language to define, raise, recognize, ignore, and handle it; for example, (ON-) condition in PL/I, exception in Ada. (I)

exception (C++ definition). (1) Any user, logic, or system error detected by a function that does not itself deal with the error but passes the error on to a handling routine (also called throwing the exception). (2) In programming languages, an abnormal situation that may arise during execution, that may cause a deviation from the normal execution sequence, and for which facilities exist in a programming language to define, raise, recognize, ignore, and handle it; for example, (ON-) condition in PL/I, exception in ADA *ISO-JTC1*.

exception handling. A type of error handling that allows control and information to be passed to an exception handler when an exception occurs. **Try** blocks, **catch** blocks and **throw** expressions are the constructs used to implement formal exception handling in C++.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators are expressions.

extension. (1) One kind of modification of an application by an enterprise; function is added without changing any function present in the application.

Note: next 2 definitions are from C++ (2) An element or function not included in the standard language.
(3) File name extension.

F

feature. (1) A choice or option available to customize the end product. A feature is built into the end product during production, not just shipped with it. A feature may be *mandatory (standard)*—that is, one of the available choices (*variants*) must be chosen; for example, engines and transmissions in cars; or *optional*—none of the variants for this feature need be chosen; for example, radios or sun roofs in cars. There are *production features*, which define allowable configurations of a given product, with associated routings and technical specifications necessary, and *sales features*, which define choices or options the customer can order. In some cases, sales and production features are identical, but in others, one sales feature may translate into several production features. See also *accessory, variant*. (2) In object-oriented programming, a collective term for methods and attributes.

filter. A command whose operation consists of reading data from standard input or a list of input files and writing data to standard output. Typically, its function is to perform some transformation on the data stream. *X/Open*.

first element. The element visited first in an iteration over a collection. Each collection has its own definition for first element. For example, the first element of a sorted set is the element with the smallest value.

framework. Pre-built class libraries that act as off-the-shelf modules for building applications to solve specific business problems.

function definition. The complete description of a function. A function definition contains an optional storage class specifier, an optional type specifier, a function declarator, parameter declarations, and a block statement (the function body).

G

generic class. See *class templates*.

given. Referring to a collection, element or function that is given as a function argument.

H

header. System-defined control information that precedes user data.

header data. The data that provides the identification and description of an object.

header file. A file that contains system-defined control information that precedes user data. Also known as an *include* file.

hierarchy (C++ definition). A structure that has different ranks or levels.

I

ID. See *somId.*

identifier (SACL definition). (1) In dialog tags, a value of 8 or fewer characters beginning with an alphabetic character (A–Z) and containing alphanumeric characters (A–Z, 0–9) in positions 2 and following. (2) One or more characters used to identify or name a data element and possibly to indicate certain properties of that data element.

identifier (C++ definition). (1) One or more characters used to identify or name a data element and possibly to indicate certain properties of that data element. *ANSI*. (2) In programming languages, a token that names a data object such as a variable, an array, a record, a subprogram, or a function. *ANSI*. (3) A sequence of letters, digits, and underscores used to identify a data object or function. *IBM*.

IDL source file. A user-written .idl file, expressed using the syntax of the *Interface Definition Language* (IDL), which describes the *interface* for a particular *class* (or classes, for a *module*). The IDL source file is processed by the *SOM Compiler* to generate the *binding files* specific to the programming languages of the class implementor and the client application. (This file may also be called the "IDL file," the "source file," or the "interface definition file.")

implementation. (Or *object implementation*.) The specification of what *instance variables* implement an *object*'s *state* and what *procedures* implement its *methods* (or *behaviors*).

implementation statement. An optional declaration within the body of the *interface* definition of a *class* in a SOM *IDL source file,* specifying information about how the class will be implemented (such as, version numbers for the class, overriding of inherited methods, or type of method resolution to be supported by particular methods). This statement is a SOM-unique statement; thus, it must be preceded by the term "#ifdef __SOMIDL__" and followed by "#endif". See also *interface declaration.*

implementation template. A template file containing *stub procedures* for *methods* that a *class* introduces or *overrides*. The implementation template is one of the *binding files* generated by the *SOM Compiler* when it processes the *IDL source file* containing class *interface declarations.* The class implementor then customizes the *implementation,* by adding language-specific code to the *method procedures*.

IMS (Information Management System). A database/data communication (DB/DC) system that can manage complex databases and networks. *IBM*.

include directive. A preprocessor directive that causes the preprocessor to replace the statement with the contents of a specified file.

incremental update. A revision to an *implementation template* file that results from reprocessing of the *IDL source file* by the *SOM Compiler*. The updated implementation file will contain new *stub procedures*, added comments, and revised *method prototypes* reflecting changes made to the *method* definitions in the IDL specification. Importantly, these updates do not disturb existing code that the class implementor has defined for the prior method procedures.

index. A list of the contents of a file or of a document, together with keys or references for locating the contents. (I) (A)

Information Management System (IMS). A database/data communication (DB/DC) system that can manage complex databases and networks. *IBM*.

inheritance (SOM definition). The technique of defining one *class* (called a *subclass, derived class,* or *child class*) as incremental differences from another class (called the *parent class, base class, superclass,* or *ancestor class*). From its parents, the subclass inherits variables and *methods* for its *instances*. The subclass can also provide additional *instance variables* and methods. Furthermore, the subclass can provide new procedures for implementing inherited methods. The subclass is then said to *override* the parent class's methods. An overriding method procedure can elect to call the parent class's *method procedure*. (Such a call is known as a *parent method call*.)

inheritance (C++ definition). An object-oriented programming technique that allows you to use existing classes as bases for creating other classes.

inheritance hierarchy. The sequential relationship from a root class to a subclass, through which the subclass inherits *instance methods, attributes,* and *instance variables* from all of its ancestors, either directly or indirectly. The root class of all SOM classes is SOMObject.

install. (1) To add a program, program option, or software to a system in such a manner that it is runnable and interacts properly with all affected programs in the system. (2) To connect hardware to a system.

installation. In system development, preparing and placing a functional unit in position for use. (T)

instance (SACL definition). A single, actual occurrence of a particular object. Any level of the object class hierarchy can have instances. For example, engineering work request number 134567 would be an instance of type engineering work request. An instance can be considered in terms of a copy of the object type frame that is filled in with particular information. The filling-in process is called *instantiation*.

instance (SOM definition). (Or *object instance* or just *object*.) A specific object, as distinguished from a *class* of objects. See also *object.*

instance (C++ definition). An object-oriented programming term synonymous with 'object'. An instance is a particular instantiation of a data type. It is simply a region of storage that contains a value or group of values. For example, if a class `box` is previously defined, two instances of a class `box` could be instantiated with the declaration:

```
box box1, box2;
```

instance data. A general term for the set of instance data variables specified for an object.

instance variable. Variables declared for access by *method procedures* of a *class*. An instance variable is declared within the body of the *implementation statement* in a SOMobjects for MVS *IDL source file*. An instance variable is “private” to the class and should not be accessed by a client program.

instantiate. To create or generate a particular instance (or object) of a data type or class of objects.

integer. A positive or negative whole number, that is, an optional sign followed by a number that does not contain a decimal place or zero.

interface. The information that a *client* must know to use a *class* namely, the names of its *attributes* and the signatures of its *methods* . The interface is described in a formal language (the *Interface Definition Language*, IDL) that is independent of the programming language used to implement the class's methods.

interface declaration. (Or *interface statement.*) The statement in the *IDL source file* that specifies the name of a new class and the names of its *parent class*(es). The “body” of the interface declaration defines the *signature* of each new *method* and any *attribute*(s) associated with the class. In SOM IDL, the body may also include an *implementation statement* (where *instance variables* are declared or a *modifier* is specified, for example to *override* a method).

Interface Definition Language (IDL). The formal language (independent of any programming language) by which the *interface* for a *class* of *objects* is defined in a .idl file, which the *SOM Compiler* then interprets to create an *implementation template* file and *binding* files. SOM's Interface Definition Language is fully compliant with standards established by the Object Management Group's Common Object Request Broker Architecture (*CORBA*).

Interface Repository (IR). The database that SOM optionally creates to provide persistent storage of objects representing the major elements of *interface* definitions. Creation and maintenance of the IR is based on information supplied in the *IDL source file.* The SOM IR Framework supports all interfaces described in the *CORBA* standard.

Interface Repository Framework. A set of *classes* that provide *methods* whereby executing programs can access the persistent objects of the *Interface Repository* to discover everything known about the programming *interfaces* of SOM classes.

interoperability. Under MVS, resources existing in parent processes do not exist in a forked child process.

J

job class. Any one of a number of job categories that can be defined. By classifying jobs and directing initiator/terminators to initiate specific classes of jobs, it is possible to control a mixture of jobs that can be performed concurrently.

K

key. A data member that is used to identify an element. A key only is a part of a complete element that can be used, for example, to establish the order of elements in a collection.

keyword (SACL definition). A parameter of a dialog tag. A keyword is specified to the left of the equal sign (=) in a keyword = attribute pair.

L

level. In a tree-structured object, the level defines the relation between the root and a particular node. The root is the highest level, nodes adjacent to the root form the next level, and so on.

library (SACL definition). A partitioned data set whose members validate data that is entered in panels. The members can also translate data entered in panels before storage in the database. Members can also translate data in the reverse order between database retrieval and display in panels.

library (C++ definition). (1) A collection of functions, calls, subroutines, or other data. *IBM*. (2) A set of object modules that can be specified in a link command.

line. A sequence of zero or more non-newline characters plus a terminating newline character. *X/Open*.

literal (SACL definition). In dialog tags, information that is coded exactly as it should appear in the panel.

literal (C++ Definition). (1) In programming languages, a lexical unit that directly represents a value; for example, 14 represents the integer fourteen, "APRIL" represents the string of characters APRIL, 3.0005E2 represents the number 300.05. *ISO-JTC1.* (2) A symbol or a quantity in a source program that is itself data, rather than a reference to data. *IBM*. (3) A character string whose value is given by the characters themselves; for example, the numeric literal 7 has the value 7, and the character literal "CHARACTERS" has the value CHARACTERS. *IBM*.

local. (1) In programming languages, pertaining to the relationship between a language object and a block such that the language object has a scope contained in that block. *ISO-JTC1*. (2) Pertaining to that which is defined and used only in one subdivision of a computer program. *ANSI*.

local variable. A variable that is only visible and usable in a single method.

location. (1) The site responsible for an object. (2) An identified place of significance to an enterprise, such as a work place, plant, or cost center. Manufacturing locations, cost-estimating locations, and engineering locations can be identified.

M

macro (SOM definition). An alias for executing a sequence of hidden instructions; in SOM, typically the means of executing a command known within a *binding file* created by the *SOM Compiler*.

macro (C++ definition). An identifier followed by arguments (may be a parenthesized list of arguments) that the preprocessor replaces with the replacement code located in a preprocessor `#define` directive.

map. An unordered flat collection that uses keys and has element equality.

mapping. In a database, the establishing of correspondences between a given logical structure and a given physical structure. (T)

member. A data object or function in a structure, union or class. Members can also be classes, enumerations, bit fields and type names.

memory leak. Occurs when dynamic memory that has been allocated to an application is not freed by the application when the memory is no longer needed.

message. (1) The primary mechanism for objects to communicate with each other. (2) Text displayed to the user to supply important information or to prompt action. Messages are defined in message members using the <MSG> and <VARSUB> dialog tags.

metaclass. A *class* whose *instances* are class objects. In SOM, any class descended from *SOMClass* is a metaclass. The *methods* a class inherits from its metaclass are called *class methods*

metaclass incompatibility. A situation where a *subclass* does not include all of the *class variables* or respond to all of the *class methods* of its *ancestor classes*. This situation can easily arise in *OOP* systems that allow programmers to explicitly specify *metaclasses*, but is not allowed to occur in SOM. Instead, SOM automatically prevents this by creating and using *derived metaclasses* whenever necessary.

method (SOM definition). A combination of a *procedure* and a name, such that many different procedures can be associated with the same name. In object-oriented programming, invoking a method on an *object* causes the object to execute a specific *method procedure*. The process of determining which method procedure to execute when a method is invoked on an object is called *method resolution*. (The *CORBA* standard uses the term “operation” for method invocation). SOM supports two different kinds of methods: static methods and dynamic methods. See also *static method* and *dynamic method.*

method (C++ definition). Method is an object-oriented programming term synonymous with member function.

method descriptor. See *descriptor*.

method ID. A number representing a zero-terminated string by which SOM uniquely represents a *method* name. See also *somId.*

method procedure. A function or procedure, written in an arbitrary programming language, that implements a *method* of a *class*. A method procedure is defined by the class implementor within the *implementation template* file generated by the *SOM Compiler.*

method resolution. The process of selecting a particular *method procedure*, given a *method* name and an object *instance*. The process results in selecting the particular function/procedure that implements the abstract method in a way appropriate for the designated object. SOM supports a variety of method-resolution mechanisms, including *offset method resolution, name-lookup resolution,* and *dispatch-function resolution*.

method table. A table of pointers to the *method procedures* that implement the *methods* that an *object* supports. See also *method token.*

method token. A value that identifies a specific *method* introduced by a *class*. A method token is used during *method resolution* to locate the *method procedure* that implements the identified method. The two basic method-resolution procedures are somResolve (which takes as arguments an *object* and a method token, and returns a pointer to a procedure that implements the identified method on the given object) and somClassResolve (which takes as arguments a *class* and a method token, and returns a pointer to a procedure that implements the identified method on an instance of the given class).

mode. A collection of attributes that specifies a file's type and its access permissions. *X/Open*. *ISO.1*.

model. A representation of a process or system that attempts to relate the most important variables in the system in such a way that analysis of the model leads to insights into the system. The model is typically used to anticipate the result of some particular strategy in the real system.

modifier. Any of a set of statements that control how a *class*, an *attribute*, or a *method* will be implemented. Modifiers can be defined in the *implementation statement* of a SOM *IDL source file*. The implementation statement is a SOM-unique extension of the *CORBA* specification. [User-defined modifiers can also be specified for use by user-written emitters or to store information in the *Interface Repository*, which can then be accessed via methods provided by the *Interface Repository Framework*.]

module (SOM definition). The organizational structure required within an *IDL source file* that contains *interface declarations* for two (or more) classes that are not a class-metaclass pair. Such *interfaces* must be grouped within a module declaration.

module (C++ definition). A program unit that usually performs a particular function or related functions, and that is distinct and identifiable with respect to compiling, combining with other units, and loading.

multibyte character. A mixture of single-byte characters from a single-byte character set and double-byte characters from a double-byte character set.

multiple inheritance (SOM definition). The situation in which a *class* is derived from (and inherits *interface* and *implementation* from) multiple parent classes.

multiple inheritance (C++ definition). An object-oriented programming technique implemented in C++ through derivation, in which the derived class inherits members from more than one base class. See also *inheritance*.

N

name space. A category used to group similar types of identifiers.

naming scope. See *scope*.

network. An arrangement of nodes and connecting branches. (T)

new. A keyword identifying a free store allocation operator. The **new** operator may be used to create class objects.

node. In a tree structure, a point at which subordinate items of data originate. *ANSI*.

NULL. In C/MVS and C++/MVS, a pointer that does not point to a data object. *IBM*.

O

object (SOM definition). (Or *object instance* or just *instance*.) An entity that has *state* (its data values) and *behavior* (its *methods*). An object is one of the elements of data and function that programs create, manipulate, pass as arguments, and so forth. An object is a way to encapsulate state and behavior. *Encapsulation* permits many aspects of the *implementation* of an object to change without affecting client programs that depend on the object's behavior. In SOM, objects are created by other objects called *classes*.

object (C++ definition). (1) A region of storage. An object is created when a variable is defined or new is invoked. An object is destroyed when it goes out of scope. (See also *instance*.) (2) In object-oriented design or programming, an abstraction consisting of data and the operations associated with that data. See also *class*. *IBM*. (3) An instance of a class.

object implementation. See *implementation.*

object-oriented programming (SACL definition). A method for structuring programs as hierarchically organized classes describing the data and operations of objects that may interact with other objects. (T)

object-oriented programming (C++ definition). A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates not on how something is accomplished but instead on what data objects comprise the problem and how they are manipulated.

object reference (SACL definition). An object that holds the identification of a persistent object in the database. The object reference can be requested to return a pointer to the persistent object. The object reference will at that time either locate the persistent object already in memory, or materialize the object from the database into memory.

object reference (SOM definition). A *CORBA* term denoting the information needed to reliably identify a particular *object*.

object request broker (ORB). See *ORB*.

OIDL. The original language used for declaring SOM *classes*. The acronym stands for Object Interface Definition Language. OIDL is still supported by SOM release 2, but it does not include the ability to specify *multiple inheritance* classes.

OOP. An acronym for “object-oriented programming.”

operating system. Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible. (T)

operation. See *method.*

operator. A symbol (such as **+**, **-**, *) that represents an operation (in this case, addition, subtraction, multiplication).

overflow. (1) That portion of an operation's result that exceeds the capacity of the intended unit of storage. (2) That portion of an operation that exceeds the capacity of the intended unit of storage. *IBM*.

override. (Or *overriding method.*) The technique by which a *class* replaces (redefines) the *implementation* of a *method* that it inherits from one of its *parent classes*. An overriding method can elect to call the parent class's *method procedure* as part of its own implementation. (Such a call is known as a *parent method call.*)

P

parameter (SACL definition). (1) A variable that is given a constant value for a specified application and that may denote the application. (I) (A) (2) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted.

parameter (C++ term). (1) In C/MVS and C++/MVS, an object declared as part of a function declaration or definition that acquires a value on entry to the function, or an identifier following the macro name in a function-like macro definition. *X/Open*. (2) Data passed between programs or procedures. *IBM*.

parameter declaration. A description of a value that a function receives. A parameter declaration determines the storage class and the data type of the value.

parent. A node that has children in a tree structure.

parent class. A *class* from which another class inherits *instance methods*, *attributes*, and *instance variables*. A parent class is sometimes called a *base class* or *superclass*.

parent method call. A technique where an *overriding method* calls the *method procedure* of its *parent class* as part of its own *implementation*.

parser. See *dialog tag parser*.

pattern. A sequence of characters used either with regular expression notation or for pathname expansion, as a means of selecting various characters strings or pathnames, respectively. The syntaxes of the two patterns are similar, but not identical. *X/Open*.

persistent object. An *object* whose *state* can be preserved beyond the termination of the *process* that created it. Typically, such objects are stored in a persistent store such as files, a database, etc..

personalize. To change an application so as to apply only to an individual's use of the application; for example, setting language and time/date formats. See also *customize*.

pointer. A variable that holds the address of a data object or function.

polymorphism (SACL definition). A property of object-oriented programming that stems from inheritance. It allows various classes in a leg of the inheritance hierarchy to be used interchangeably by parent classes. A class is polymorphic with its parents if it does not redefine the parameters or external behavior of its parent's methods.

polymorphism (SOM definition). An object-oriented programming feature where a method name can denote more than one *method procedure.* For example, when a SOM class *overrides* a *parent class* definition of a method to change its behavior. The term literally means “having many forms.”

polymorphism (C++ definition). The technique of taking an abstract view of an object or function and using any concrete objects or arguments that are derived from this abstract view.

precedence. The priority system for grouping different types of operators with their operands.

preprocessor. A phase of the compiler that examines the source program for preprocessor statements that are then executed, resulting in the alteration of the source program.

primitive. A data type that is supported by the C programming language. A primitive cannot receive message calls.

principal. The user on whose behalf a particular (remote) *method* call is being performed.

private. Pertaining to a class member that is only accessible to member functions and friends of that class.

private IDL. That part of an IDL specification that a class builder has identified as being hidden from a user.

procedure. A small section of code that executes a limited, well-understood task when called from another program. In SOM, a *method procedure* is often referred to as a procedure. See also *method procedure.*

process (SACL definition). A series of instructions (a program or part of a program) that a computer executes in a multitasking environment.

process (C++ definition). (1) An instance of an executing application and the resources it uses. (2) An address space and single thread of control that executes within that address space, and its required system resources. A process is created by another process issuing the fork() function. The process that issues the fork() function is known as the parent process, and the new process created by the fork() function is known as the child process. *X/Open*. *ISO.1*.

process ID. The unique identifier representing a process. A process ID is a positive integer. (Under ISO only, it is a positive integer *that can be contained in a pid_t*.) A process ID will not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID will not be reused by the system until the process group lifetime ends. A process that is not a system process will not have a process ID of 1. *X/Open*. *ISO.1*.

protected. A protected member of a class is accessible to member functions and friends of that class, or member functions and friends of classes derived from that class.

prototype. A function declaration or definition that includes both the return type of the function and the types of its parameters. See *function prototype*.

public. A public member of a class is accessible to all functions.

public IDL. That part of an IDL specification that a class builder has identified as being accessible by a user.

Q

qualified name. Used to qualify a nonclass type name such as a member by its class name.

queue. A sequence with restricted access in which elements can only be added at the back end (or bottom) and removed from the front end (or top). A queue is characterized by first-in, first-out behavior and chronological order.

R

receiver. See *target object.*

relationship. A definition of the properties that exist between two entities.

replica. When an object is replicated among a set of processes (using the Replication Framework), each process is said to have a replica of the object. From the view point of any application model, the replicas together represent a single object.

replicated object. An *object* for which *replicas* (copies) exist. See *replica.*

return code. A value returned to a program to indicate the results of an operation requested by that program.

root. (1) A node that has no parent. All other nodes of a tree are descendants of the root. (2) In the AIX operating system, the user name for the system user with the most authority. *IBM.* (3) In the OS/2 operating system, the base directory.

S

S-name. An external non-C++ name in an object module produced by compiling with the NOLONGNAME option. Such a name is up to 8 characters long and single case.

scope (SOM definition). (Or *naming scope.*) That portion of a program within which an identifier name has “visibility” and denotes a unique variable. In SOM, an *IDL source file* forms a scope. An identifier can only be defined once within a scope; identifiers can be redefined within a nested scope. In a .idl file, modules, interface statements, structures, unions, methods, and exceptions form nested scopes.

scope (C++ definition). (1) That part of a source program in which a variable is visible. (2) That part of a source program in which an object is defined and recognized.

sequence. A sequentially ordered flat collection.

services. Routines complementary to the application transaction support.

set. An unordered flat collection with element equality.

shadowing. In the Emitter Framework, a technique that is required when any of the *entry classes* are subclassed. Shadowing causes instances of the new subclass(es) (rather than instances of the original entry classes) to be used as input for building the object graph, without requiring a recompile of emitter framework code. Shadowing is accomplished by using the macro SOM_SubstituteClass.

signal. (1) A condition that may or may not be reported during program execution. For example, SIGFPE is the signal used to represent erroneous arithmetic operations such as a division by zero. (2) A mechanism by which a process may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term *signal* is also used to refer to the event itself. *X/Open*. *ISO.1*. (3) In AIX operating system operations, a method of interprocess communication that simulates software interrupts. *IBM*.

signature. The collection of types associated with a *method* (the type of its return value, if any, as well as the number, order, and type of each of its arguments).

Sockets class. A class that provides a common communications interface to Distributed SOM, the Replication Framework, and the Event Management Framework. The Sockets class provides the base interfaces (patterned after TCP/IP sockets); the *subclasses* TCPIPSockets, NBSockets, and IPXSockets provide actual implementations for TCP/IP, Netbios, and Netware IPX/SPX, respectively.

SOM Compiler. A tool provided by the SOM Toolkit that takes as input the interface definition file for a class (the .idl file) and produces a set of *binding files* that make it more convenient to implement and use SOM classes.

SOMClass. One of the three primitive *class objects* of the SOM run-time environment. SOMClass is the root (meta)class from which all subsequent *metaclasses* are derived. SOMClass defines the essential *behavior* common to all SOM *class objects.*

SOMClassMgr. One of the three primitive *class objects* of the SOM run-time environment. During SOM initialization, a single *instance* (*object*) of SOMClassMgr is created, called SOMClassMgrObject. This object maintains a directory of all SOM classes that exist within the current process, and it assists with dynamic loading and unloading of class libraries.

SOM-derived metaclass. See *derived metaclass.*

somId. A pointer to a number that uniquely represents a zero-terminated string. Such pointers are declared as type somId. In SOM, somId's are used to represent *method* names, *class* names, and so forth.

SOMObject. One of the three primitive *class objects* of the SOM run-time environment. SOMObject is the root class for all SOM (sub)classes. SOMObject defines the essential *behavior* common to all SOM *objects.*

somSelf. Within *method procedures* in the *implementation* file for a class, a parameter pointing to the *target object* that is an *instance* of the *class* being implemented. It is local to the *method procedure.*

somThis. Within *method procedures,* a local variable that points to a data structure containing the *instance variables* introduced by the *class*. If no instance variables are specified in the SOM *IDL source file*, then the somThis assignment statement is commented out by the *SOM Compiler*.

stack. A sequence with restricted access in which elements are added to the top and removed from the top. A stack is characterized by last-in, first-out behavior and reverse chronological order.

standard output. (1) An output stream usually intended to be used for primary data output. *X/Open*. (2) In the AIX operating system, the primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can go to a file or to another command. *IBM*.

state (of an object). The data (*attributes, instance variables* and their values) associated with an *object.* See also *behavior.*

static. A keyword used for defining the scope and linkage of variables and functions. For internal variables, the variable has block scope and retains its value between function calls. For external values, the variable has file scope and retains its value within the source file. For class variables, the variable is shared by all objects of the class and retains its value within the entire program.

status. The condition or situation of an object. The status of an object determines the activities that may be performed on that object.

status code. Indicates the status of an object, for example, created, distributed, recalled, deleted, etc.

stream (SACL definition). An object that is a linked-list structure of stream elements. Streams are used to represent unbounded lists.

string. In dialog tags, a series of characters processed as a single item. Strings with multiple words require quotation marks around them and are referred to as *quoted strings*.

structure. A construct (a class data type) that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types. A structure can be used in all places a class is used. The initial projection is public.

stub procedures. *Method procedures* in the *implementation template* generated by the *SOM Compiler*. They are procedures whose bodies are largely vacuous, to be filled in by the implementor.

subclass. A *class* that inherits *instance methods*, *attributes*, and *instance variables* directly from another class, called the *parent class, base class*, *superclass,* or indirectly from an *ancestor class*. A subclass may also be called a *child class* or *derived class.*

subclassing. The process whereby a new *class*, as it is created (or *derived*), inherits *instance methods, attributes,* and *instance variables* from one or more previously defined *ancestor classes*. The immediate *parent class(es)* of a new class must be specified in the class's *interface declaration.* See also *inheritance.*

subset. Given two sets A and B, B is a subset of A if and only if all elements of B are also elements of A.

suffix. The part of a dialog tag that follows the > in the tag. Suffixes are usually used to code literal information.

switch statement. A C++ language statement that causes control to be transferred to one of several statements depending on the value of an expression.

symbol. In the Emitter Framework, any of a (standard or user-defined) set of names (such as, className) that are used as placeholders when building a text template to pattern the desired *emitter* output. When a template is emitted, the symbols are replaced with their corresponding values from the emitter's symbol table. Other symbols (such as, classSN) have values that are used by section-emitting methods to identify major sections of the template (which are correspondingly labeled as “classS” or by a user-defined name).

T

table. An array of data each item of which can be unambiguously identified by means of one or more arguments.

target object. (Or *receiver.*) The object responding to a *method* call. The target object is always the first formal parameter of a *method procedure*. For SOM's C-language bindings, the target object is the first argument provided to the method invocation macro, *_methodName*.

task. (1) In a multiprogramming or multiprocessing environment, one or more sequences of instructions treated by a control program as an element of work to be accomplished by a computer. *ISO-JTC1. ANSI.* (2) A routine that is used to simulate the operation of programs. Tasks are said to be *nonpreemptive* because only a single task is executing at any one time. Tasks are said to be *lightweight* because less time and space are required to create a task than a true operating system process.

template. A family of classes or functions with variable types.

template class. A class instance generated by a class template.

this. A keyword that identifies a special type of pointer that references in a member function the class object with which the member function was invoked.

type. The description of the data and the operations that can be performed on or by the data. Also see *data type*.

type code. A code that identifies the kind of object. Type codes for routings include conceptual, master, and individual.

type definition. A definition of a data type.

U

union. (1) In C/MVS or C++/MVS, a variable that can hold any one of several data types, but only one data type at a time. *IBM*. (2) For bags, there is an additional rule for duplicates: If bag P contains an element *m* times and bag Q contains the same element *n* times, then the union of P and Q contains that element *m*+*n* times.

usage bindings. The language-specific *binding* files for a *class* that are generated by the *SOM Compiler* for inclusion in client programs using the class.

using object.

V

value logging. In the Replication Framework, a technique for maintaining consistency among *replicas* of a replicated object, whereby the new value of the object is distributed after the execution of a method that updates the object.

variable. In SOMobjects for MVS, the data that defines a class of objects and that holds (or contains) a value in an instance of that class. A data variable can define a class and its instances (objects) or a metaclass and its instances (class objects). The are sometimes referred to, respectively, as instance variables and class variables

variable value. In SOMobjects for MVS, the data that is contained in an instantiation of a data variable as a program is executing.

variant. One of the allowable choices for an accessory or a feature. See also *accessory, feature*.

W

warning message. In SAA Common User Access architecture, a message that tells a user that a requested action has been suspended because something undesirable could occur. A user can continue the requested action, withdraw the requested action, or get help.

white space. (1) Space characters, tab characters, form-feed characters, and new-line characters. (2) A sequence of one or more characters that belong to the space character class as defined via the LC_CTYPE category in the current locale. In the POSIX locale, white space consists of one or more blank characters (space and tab characters), newline characters, carriage-return characters, form-feed characters, and vertical-tab characters. *X/Open*.

+0 (—0). An algebraic sign provides additional information about any variable that has the value zero. Although all precisions have distinct representations for +0, -0, +Inf, and -Inf, the signs are significant in some circumstances, such as division by zero, and not in others. *X/Open*.

Index

Communicating Your Comments to IBM

OS/390
SOMobjects
Messages, Codes and Diagnosis

Publication No. SC28-1996-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing an RCF from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - FAX: (International Access Code)+1+914+432-9405
- If you prefer to send comments electronically, use this network ID:
  - IBMLink: (United States customers only): KGNVMC(MHVRCS)
  - IBM Mail Exchange: USIB6TC9 at IBMMAIL
  - Internet e-mail: mhvrcfs@vnet.ibm.com
  - World Wide Web: http://www.s390.ibm.com/os390

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

Reader's Comments — We'd Like to Hear from You

OS/390
SOMobjects
Messages, Codes and Diagnosis

Publication No. SC28-1996-01

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: ____________________________________

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ]	As an introduction	[ ]	As a text (student)
[ ]	As a reference manual	[ ]	As a text (instructor)
[ ]	For another purpose (explain)		

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Name	Address
Company or Organization	
Phone No.	

Reader's Comments — We'd Like to Hear from You
SC28-1996-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie NY 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

SC28-1996-01

Cut or Fold
Along Line

Program Number: 5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC28-1996-01