OS/390

SOMobjects Getting Started

GA22-7248-01

OS/390

SOMobjects Getting Started

GA22-7248-01

Note

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

Second Edition, September 1997

| This edition applies to OS/390 Version 2 Release 4 (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 e-mail: mhvrcfs@vnet.ibm.com
World Wide Web: http://www.s390.ibm.com/os390

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

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM	IBMLink	OpenEdition
OS/390	RACF	SOMobjects

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About This Book

This book provides an overview of how you can use OS/390 SOMobjects. The first chapter describes how to run the class and application samples that are provided with OS/390 SOMobjects. The following chapters then use these examples to take you through each step in more detail. Finally, reference information is provided to help you find more information in the other books in the OS/390 SOMobjects library.

Who Should Use This Book

This book is intended to give application programmers an overview of how to create and implement classes with OS/390 SOMobjects. Some familiarity with the concepts of object-oriented programming and the OS/390 operating system is assumed.

How This Book is Organized

This book contains the following sections:

- A checklist to provide a brief overview of how to use OS/390 SOMobjects
- Information to create new classes with OS/390 SOMobjects
- Information to create an application program to implement the classes
- Information to use classes and applications in a distributed environment
- An overview of the OS/390 SOMobjects library and a summary of the sample data sets provided.

Where to Find More Information

Table 1 lists other publications that provide more information about how to use OS/390 SOMobjects. These books include additional examples that demonstrate other methods for creating and implementing classes with OS/390 SOMobjects. For more information about these books, see “OS/390 SOMobjects Library Reference” on page 29.

Table 1. OS/390 SOMobjects Publications

Title	Order Number
OS/390 V2R4.0 SOMobjects Configuration and Administration Guide	GC28-1851
OS/390 V2R4.0 SOMobjects Programmer's Guide	GC28-1859
OS/390 V2R4.0 SOMobjects Object Services	SC28-1995
OS/390 V2R4.0 SOMobjects Messages, Codes, and Diagnosis	SC28-1996
OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1	SC28-1997
OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 2	SC28-1998
OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 3	SC28-1999

Chapter 1. Quick Start Checklist

The following sections describe the essential steps for building new classes and running object-oriented applications with OS/390 SOMobjects. It also describes how to run the sample files provided with OS/390 SOMobjects.

Each step is explained in more detail in the chapters that follow. For more information, you can also refer to the *OS/390 V2R4.0 SOMobjects Programmer's Guide* and the *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

Building Non-distributed Applications

The process of building a non-distributed object application generally consists of the following steps:

- Build the classes
  - __ 1. Determine the structure and characteristics of each class.
  - __ 2. Write IDL code to define the interface for each class you will implement.
  - __ 3. Compile the IDL with the OS/390 SOMobjects compiler.
  - __ 4. Write the implementation for each class in a supported language, such as C/C++ for OS/390 or COBOL.
  - __ 5. Compile the implementation code.
  - __ 6. Prelink and link the class implementation.
- Build the application that will use the classes
  - __ 1. Determine the structure of the application.
  - __ 2. Write the application program in a supported language, such as C/C++ for OS/390 or COBOL.
  - __ 3. Compile the application.
  - __ 4. Prelink and link the application.
  - __ 5. Run the application.

Running the Non-distributed Sample

OS/390 SOMobjects provides the SOMMVS.SGOSJCL(GOS1AUTO) sample data set to help you build and run your first non-distributed object oriented application. Follow the instructions in this sample data set to build a class, Vehicle, and a subclass, Car; this sample also builds and runs the DRVCAR application, which uses those classes.

Note: Before you can run this sample, OS/390 SOMobjects must be configured for non-distributed applications and running on your system. These tasks are generally performed by your system programmer. See the *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* for more information.

Building Distributed Applications

The process of building a distributed object application generally consists of the following steps:

- Build the server implementation
  - __ 1. Determine the structure and characteristics of each class.
  - __ 2. Write IDL code to define the interface for each class you will implement.
  - __ 3. Compile the IDL with the OS/390 SOMobjects compiler.
  - __ 4. Write the implementation for each class in a supported language, such as C/C++ for OS/390 or COBOL.
  - __ 5. Compile the implementation code.
  - __ 6. Prelink and link the class implementation.
- Register the class and server implementations
  - __ 1. Register the application server (one-time procedure for each server)
    - __ a. Define the server to the OS/390 Workload Manager (WLM).
    - __ b. Establish security definitions for your server.
    - __ c. Use the REGIMPL utility to register each server with the OS/390 SOMobjects Subsystem.
  - __ 2. Register the class (required for each class)
    - __ a. Use the REGIMPL utility to register each class with the OS/390 SOMobjects Subsystem.
- Build the client application
  - __ 1. Determine the structure of the application.
  - __ 2. Write the application program in a supported language.
  - __ 3. Compile the application.
  - __ 4. Prelink and link the application.
  - __ 5. Run the application.

Building the Distributed Samples

To build your first distributed object oriented application, follow the instructions in the sample data sets provided with OS/390 SOMobjects:

- SOMMVS.SGOSJCL(GOSSMPSC)
- SOMMVS.SGOSJCL(GOSSMPCC)
- SOMMVS.SGOSJCL(GOSSMPLK)

These JCL samples build and link the resulting object code for the distributed versions of the Vehicle and Car classes. These samples also contain JCL for several other distributed examples that are provided with OS/390 SOMobjects.

Note: Before you can run these samples, OS/390 SOMobjects must be configured for distributed applications and running on your system. The class and server implementations must also be registered with OS/390 SOMobjects. These tasks are generally performed by your system programmer. See the *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* for more information.

Chapter 2. Introducing OS/390 SOMobjects

This section provides a high-level overview of object-oriented concepts and how those concepts are implemented with OS/390 SOMobjects. If you are already familiar with these general concepts but would like more information about how to use OS/390 SOMobjects, you should proceed to Chapter 3, “Building the Classes” on page 7.

Understanding OS/390 SOMobjects

Traditional programming techniques focus on the structure of programming instructions; you use a specific programming language to define the procedures to complete a specific set of tasks. With object-oriented programming, however, the focus shifts to identifying the type of data that is needed to perform the task. Instructions are then written to interact with the characteristics that are associated with that data.

In object-oriented programming, a *class* is a template that is used to create an *instance* (or *instantiation*) of an object. A class contains information that defines the state of an object (its characteristics or *attributes*). A class also defines the operations (*methods*) that can be performed on the object.

After a class has been defined, it can be used as the basis for another object that may have related qualities. This related class *inherits* the characteristics of the first class; it can also define other characteristics that are unique to itself. This original class is often referred to as the *parent class*; the second class is often called a *child* or *subclass*.

The System Object Model (SOM) is an object-oriented programming technology for building classes, organizing class libraries, and manipulating class libraries. OS/390 SOMobjects is IBM's implementation of the Common Object Request Broker Archictecture (CORBA) for the OS/390 environment. OS/390 SOMobjects is comprised of the following parts:

- OS/390 SOMobjects Runtime Library
- OS/390 SOMobjects Application Development Environment

With OS/390 SOMobjects, you can shift your application programming focus to the object-oriented approach. You can write your classes and the application programs that access them in a programming language, such as OS/390 C/C++ or COBOL for MVS. You can build a class library in one language and write the application program that uses the class in another language. Applications can be readily ported for use in other environments. And, the concept of inheritance helps to promote code reuse and improved quality, as you can build new applications from existing, previously-tested components.

OS/390 SOMobjects Runtime Library

The OS/390 SOMobjects Runtime Library is a base element of the OS/390 operating system. The OS/390 SOMobjects Runtime Library provides the following set of services to help you run object-oriented applications on your system:

- Executable code for the SOM Kernel and several frameworks
- A file containing interface information about the SOM Kernel and frameworks and a utility (IRDUMP) to view the information in the file
- Support for vendor- and user-written frameworks.

OS/390 SOMobjects Kernel

The OS/390 SOMobjects Kernel is a component of the OS/390 SOMobjects Runtime Library. The SOM Kernel defines the object classes that provide the basis for other classes. When the OS/390 SOMobjects environment initializes, several basic objects are created (SOMObject, SOMClass, and SOMClassMgr). OS/390 SOMobjects provides a rich set of methods for use with each of these basic objects; see the *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* for more information.

The SOMObject class is the parent class for all other classes. That is, it defines characteristics and behaviors that are essential to all OS/390 SOMobjects objects and classes. Figure 2 on page 8 shows the statements you would use to access the SOMObject information when defining a new class. For more information about the OS/390 SOMobjects Kernel and the object classes, see the *OS/390 V2R4.0 SOMobjects Programmer's Guide*.

Frameworks

OS/390 SOMobjects also provides several special class libraries, called frameworks, that you can use to enhance your application programs. For example, the Interface Repository (IR) Framework provides run-time access to all of the information contained in the IDL description of a class of objects. Another framework, the Emitter Framework, lets you alter the format of the output files produces by the OS/390 SOMobjects compiler.

One framework, called Distributed SOMobjects (or DSOM), lets applications access classes and objects that reside in different address spaces, processes, or systems. Distributed SOMobjects gives your applications transparent access to remote objects. Chapter 5, “Running in a Distributed Environment” on page 19 provides an overview of how to create classes and an client application for use in the distributed environment. See the *OS/390 V2R4.0 SOMobjects Programmer's Guide* for more information about Distributed SOMobjects and the other OS/390 SOMobjects frameworks.

OS/390 SOMobjects Application Development Environment

The OS/390 SOMobjects Application Development Environment is an optional feature of OS/390. Using OS/390 SOMobjects Application Development Environment, you can extend the class libraries provided with OS/390 SOMobjects or define new classes for your applications. The OS/390 SOMobjects Application Development Environment provides the following services to help you develop object-oriented applications:

- OS/390 SOMobjects Compiler
- IDL files for the OS/390 SOMobjects classes and supported Frameworks
- Utilities (PDL and NEWEMIT) to help you create public versions of IDL and specialized OS/390 SOMobjects Compiler code generators, called emitters.

OS/390 SOMobjects Compiler

Using the OS/390 SOMobjects Compiler, you can create and implement new classes for your applications. The OS/390 SOMobjects Compiler processes the Interface Definition Language (IDL) source code. IDL is a language that describes the characteristics (attributes and methods) of a class. The IDL source code defines the interface to the class.

The OS/390 SOMobjects Compiler also produces information to help you implement the class in a particular language. Depending on the options you specify, the OS/390 SOMobjects compiler can produce the following files, which are used when you compile the classes into an object module for a specific language:

- Implementation template, which contains code that is specific to your application and the language in which it is written
- Implementation header binding, which is included in the implementation template
- Client program header binding, which is included in application programs that will implement the defined class.

You can use several methods to invoke the OS/390 SOMobjects compiler. For example, you can use ISPF panels (if they are installed on your system) to issue compiler commands. You can also issue commands from the TSO command line, the OS/390 OpenEdition shell, and a REXX exec; or, you can use JCL statements to run the job in batch mode. OS/390 SOMobjects provides examples that use OpenEdition scripts, REXX execs, and JCL statements; the examples shown in this book use JCL statements to run the OS/390 SOMobjects compiler. Page 9 demonstrates how you can use the OS/390 SOMobjects compiler. See Figure 4 on page 9 and the SOMMVS.SGOSJCL(GOS1AUTO) data set for more information about these JCL statements.

For more information about the OS/390 SOMobjects compiler and other features provide with the OS/390 SOMobjects Application Development Environment, see the *OS/390 V2R4.0 SOMobjects Programmer's Guide*.

Chapter 3. Building the Classes

This chapter takes you through the basic steps of building new classes in a non-distributed environment. The examples shown in this section are provided as samples with OS/390 SOMobjects; see Table 4 on page 31 for a summary of these sample data set names.

Step 1: Determining What to Use

When building new classes, you identify the attributes and methods that are needed to define a particular object. You can then identify the characteristics that can be inherited from another class. You should also consider the language in which you will implement the classes. This information helps you determine the options you will need to specify when compiling the code.

The following sections demonstrate how the Vehicle and Car classes, which OS/390 SOMobjects provides as code samples, are built. Both classes are implemented in the C programming language. You can use these examples as the basis for creating your own classes. Figure 1 shows the relationship between the SOMObject class and the Vehicle and Car classes.

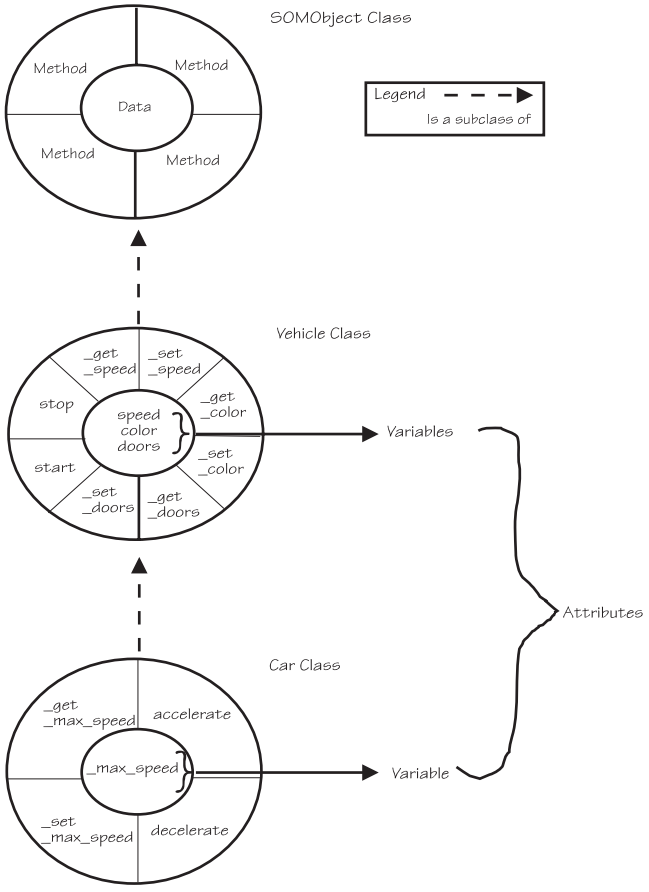

Figure 1. Relationship between the SOMObject, Vehicle, and Car Classes

Step 2: Creating the Source IDL for the Classes

This section focuses on the IDL code that defines the Vehicle and Car classes. To create a new class, you first need to write IDL code to define the attributes and methods associated with the class. When you define an attribute and compile the IDL, OS/390 SOMobjects automatically creates two additional methods called *accessor* methods. One accessor retrieves the value of an attribute (_get_) and the other sets the value (_set_) of an attribute.

Defining the Vehicle Class

Figure 2 shows the IDL code that defines a new class called “Vehicle.” (This code is included in the SOMMVS.SGOSSMPI.IDL(VEHICLE) data set.) The Vehicle class defines two methods, start and stop, that are common actions for some type of vehicle. The Vehicle class also defines attributes that are common to many types of vehicles: speed, number of doors, and a color.

The first two statements in Figure 2 enable OS/390 SOMobjects to create an instance of the Vehicle class. As noted on page 4, the SOMObject class defines the basic characteristics of all classes. The final statements enable OS/390 SOMobjects to create the accessor methods for each attribute.

```
#include <somobj.idl>
interface Vehicle : SOMObject
{
 /* Vehicle attributes */
 attribute short speed;
 attribute short doors;
 attribute string color;

 /* The Start function */
 void start();

 /* The Stop function  */
 void stop();

 implementation
 {
  filestem = vehicle;
  functionprefix = vehicle;
  dllname = vehicle;
  releaseorder: _get_speed, _set_speed,
         _get_color, _set_color,
         _get_doors, _set_doors,
         start, stop;
 };
};
```

Figure 2. Vehicle — IDL Code Definition

Defining the Car Class

Figure 3 on page 9 shows the source IDL code that defines a class called “Car.” (This code is included in the SOMMVS.SGOSSMPI.IDL (CAR) data set.) In this example, Car represents a specific type of a vehicle; it inherits the characteristics defined in the Vehicle class. Car defines the max_speed attribute and the related accessor methods; it also defines two new methods: accelerate and decelerate.

```
#include <vehicle.idl>
interface Car : Vehicle
{
 /* Car attributes */
 attribute long max_speed;

 /* The Accelerate method */
 void accelerate(in long mSpeed);   /* Takes maximum speed from caller */

 /* The Decelerate function  */
 void decelerate();

 implementation
 {
  filestem = car;
  functionprefix = Car;
  dllname = car;
  releaseorder: _set_max_speed, _get_max_speed,
         accelerate, decelerate;
 };
};
```

Figure 3. Car — IDL Code Definition

Step 3: Running the OS/390 SOMobjects Compiler

After creating the IDL for the classes, you need to run the OS/390 SOMobjects compiler. The compiler produces files to help you compile the classes into an object module for a specific language. Figure 4 shows the JCL statements to run the OS/390 SOMobjects compiler as a batch job; these statements are included in the SOMMVS.SGOSJCL(GOS1AUTO) data set. Because the `-sh:ih` value is specified on the SCPARMS= parameter, the OS/390 SOMobjects compiler produces C binding files.

```
//GOS1AUTO JOB <JOB CARD PARAMETERS >
//  SET SOM=SOMMVS
//  SET LE=CEE
//  SET CCXX=CBC
//  SET IDLPRFX=SOMMVS.SGOSSMPI
//  SET HDSN=SOMMVS.SGOSSMPH.H
//  SET CDSN=SOMMVS.SGOSSMPC.C
/*JOBPARM  T=1,L=50
//ORDER JCLLIB ORDER=(&CCXX..SCBCPRC,&LE..SCEEPROC)
//SC       PROC INDSN=,
//         MEM=,
//         SCPARMS='',
//         SOMPRFX=&SOM.
//SOMC     EXEC PGM=SC,REGION=40M,
//   PARM=('&SCPARMS ''&INDSN.(&MEM.)''')
//STEPLIB  DD   DSN=&SOMPRFX..SGOSLOAD,DISP=SHR
//         DD   DSN=&LE..SCEERUN,DISP=SHR
//SOMENV   DD   DSN=&SOMPRFX..SGOSPROF(GOSENV),DISP=SHR
//SYSPRINT DD   SYSOUT=*
//        PEND
//*
//SCCAR    EXEC SC,MEM=CAR,
//         INDSN=&IDLPRFX..IDL,
//         SCPARMS='-sh:ih'
//SCVEHIC  EXEC SC,MEM=VEHICLE,
//         INDSN=&IDLPRFX..IDL,
//         SCPARMS='-sh:ih'
```

Figure 4. JCL Statements to Run the OS/390 SOMobjects Compiler

Note: The JCL provided in the SOMMVS.SGOSJCL(GOS1AUTO) data set does not produce C implementation templates because OS/390 SOMobjects provides these templates as examples. To generate a C implementation template when you create other classes, specify the `-sc:h:ih` OS/390 SOMobjects compiler option.

Reviewing the Implementation Templates

The implementation template, which is created by the OS/390 SOMobjects compiler, contains the basic information you need to implement the class in a particular language. In these examples, the templates are created for C.

For example, Table 2 compares the IDL for Vehicle to the C template produced by the OS/390 SOMobjects compiler. In the Vehicle IDL, lines **1a** and **2a** define the start and stop methods for the class. In the implementation template, lines **1b** and **2b** show the basic structure of how each method is defined in C.

Table 2. Vehicle — Comparison of IDL and C Implementation Template

IDL Code

```
   #include <somobj.idl>

   interface Vehicle : SOMObject
   {

    /* Vehicle attributes */
    attribute short speed;
    attribute short doors;
    attribute string color;

    /* The Start function */
1a   void start();

    /* The Stop function  */
2a   void stop();

    implementation
    {
     filestem = vehicle;
     functionprefix = vehicle;
     dllname = vehicle;
     releaseorder: _get_speed, _set_speed,
            _get_color, _set_color,
            _get_doors, _set_doors,
            start, stop;
    };
   };
```

C Implementation Template

```
   ??=ifdef __COMPILER_VER__
      ??=pragma filetag ("IBM-1047")
   ??=endif

   #pragma nosequence nomargins
   #ifndef _MVS
      #define _MVS
   #endif

   #ifndef SOM_Module_vehicle_Source
   #define SOM_Module_vehicle_Source
   #endif
   #define Vehicle_Class_Source

   #include <dd:ih(vehicle)>

1b SOM_Scope void SOMLINK vehiclestart(Vehicle somSelf,
                                       Environment *ev)
   {
       VehicleData *somThis = VehicleGetData(somSelf);
       VehicleMethodDebug("Vehicle","vehiclestart");
   }

2b SOM_Scope void  SOMLINK vehiclestop(Vehicle somSelf,
                                       Environment *ev)
   {
       VehicleData *somThis = VehicleGetData(somSelf);
       VehicleMethodDebug("Vehicle","vehiclestop");
   }
```

Similarly, Table 3 on page 11 shows the IDL for Car and the corresponding C template that is produced by the OS/390 SOMobjects compiler. (OS/390 SOMobjects provides this template in the SOMMVS.SGOSSMPC.C(CAR) data set.) Lines **3a** and **4a** of the Car IDL define the accelerate and decelerate methods. In the implementation template, lines **3b** and **4b** show the basic structure of how those methods are defined in C.

Table 3. Car — Comparison of IDL and C Implementation Template

IDL Code

```
   /* The Car class */
   #include <vehicle.idl>

   interface Car : Vehicle
   {

    /* Car attributes */
    attribute long max_speed;

    /* The Accelerate method */
    /* Takes maximum speed from caller */
3a   void accelerate(in long mSpeed);

   /* The Decelerate function  */
4a   void decelerate();

    implementation
    {
     filestem = car;
     functionprefix = Car;
     dllname = car;
     releaseorder: _set_max_speed, _get_max_speed,
            accelerate, decelerate;
    };
   };
```

C Implementation Template

```
   ??=ifdef __COMPILER_VER__
      ??=pragma filetag ("IBM-1047")
   ??=endif

   #pragma nosequence nomargins
   #ifndef _MVS
      #define _MVS
   #endif

   /*  This file was generated by the SOM Compiler
    *  and Emitter Framework.
    *  Generated using template emitter:
    *      SOM Emitter emitctm: 2.23.1.10
    */

   #ifndef SOM_Module_car_Source
   #define SOM_Module_car_Source
   #endif
   #define Car_Class_Source

   #include <dd:ih(car)>

3b SOM_Scope void  SOMLINK Caraccelerate(Car somSelf,
                                         Environment *ev,
                                         long mSpeed)
   {
       CarData *somThis = CarGetData(somSelf);
       CarMethodDebug("Car","Caraccelerate");
   }

4b  SOM_Scope void  SOMLINK Cardecelerate(Car somSelf,
                                         Environment *ev)
   {
       CarData *somThis = CarGetData(somSelf);
       CarMethodDebug("Car","Cardecelerate");
   }
```

Step 4: Updating the Implementation Templates

The OS/390 SOMobjects compiler produces implementation templates that are designed to show you how to implement a class in a particular language. Before you can use the class, you may need to update the template with the appropriate code to implement the methods associated with the class.

Updating the Vehicle Template

As Figure 5 on page 12 shows, you can update the implementation template to produce a message when the vehicle is started or stopped. Lines **1c** and **2c** show the updates made to the procedures that were created in the implementation template (lines **1b** and **2b**) shown in Table 2 on page 10. In this example, code has been added to print a message when the vehicle starts and stops. OS/390 SOMobjects provides this code in the SOMMVS.SGOSSMPC.C(VEHICLE) data set.

```
??=ifdef __COMPILER_VER__
   ??=pragma filetag ("IBM-1047")
??=endif

#pragma nosequence nomargins
#ifndef _MVS
   #define _MVS
#endif

#ifndef SOM_Module_vehicle_Source
#define SOM_Module_vehicle_Source
#endif
#define Vehicle_Class_Source

#include <dd:ih(vehicle)>

SOM_Scope void  SOMLINK vehiclestart(Vehicle somSelf,  Environment *ev)
{
    VehicleData *somThis = VehicleGetData(somSelf);
    VehicleMethodDebug("Vehicle","vehiclestart");
1c  _speed = 0;
    printf("The vehicle is started.  The engine is running idle\n");
}

SOM_Scope void  SOMLINK vehiclestop(Vehicle somSelf,  Environment *ev)
{
    VehicleData *somThis = VehicleGetData(somSelf);
    VehicleMethodDebug("Vehicle","vehiclestop");

2c  printf("The vehicle is stopped.  The engine is off\n");
}
```

Figure 5. Vehicle — Updated C Template

Updating the Car Template

Figure 6 shows the updated C template for Car. This sample is provided in the SOMMVS.SGOSSMPC.C(CAR) data set. Lines **3c** and **4c** show the start of the updates that were made to the procedures (**3b** and **4b**) created in the initialization template shown in Table 3 on page 11. Code is added to create a loop to accelerate and decelerate the vehicle and to produce status messages.

```
??=ifdef __COMPILER_VER__
   ??=pragma filetag ("IBM-1047")
??=endif

#pragma nosequence nomargins
#ifndef _MVS
   #define _MVS
#endif

#ifndef SOM_Module_car_Source
#define SOM_Module_car_Source
#endif
#define Car_Class_Source

#include <dd:ih(car)>

/* Take target speed from caller
 * The accelerate function       */

SOM_Scope void  SOMLINK Caraccelerate(Car somSelf, Environment *ev, long mSpeed)
```

Figure 6 (Part 1 of 2). Car — Updated C Template

```
    {
3c      int i;             /* Loop control variable */
        long targetspeed;  /* Target speed to accelerate to */
        CarData *somThis = CarGetData(somSelf);
        CarMethodDebug("Car","Caraccelerate");
             targetspeed = mSpeed;
        if (_max_speed < targetspeed)
         {
          printf("Requested speed was %i MPH \n", mSpeed) ;
          printf("Cannot exceed the maximum speed of %i MPH \n", _max_speed);
          targetspeed = _max_speed ;
         }
        printf("Accelerating to %i MPH \n", targetspeed);

        /* Show acceleration up to the desired speed */
        for (i = Vehicle__get_speed(somSelf, ev); i <= targetspeed; i++)
         {
          printf("Current speed = %i MPH \n", i);
         }
       /* Save the current speed in the speed instance variable */
       Vehicle__set_speed(somSelf, ev, targetspeed);
    }
    SOM_Scope void  SOMLINK Cardecelerate(Car somSelf,  Environment *ev)
    {
        int i;             /* Loop control variable */
        CarData *somThis = CarGetData(somSelf);
        CarMethodDebug("Car","Cardecelerate");
4c      printf("Decelerating\n");

        for (i = Vehicle__get_speed(somSelf, ev); i >= 0; i--)
         {
          printf("Current speed = %i MPH \n", i);
         }
      /* Save the current speed in the speed instance variable */
      Vehicle__set_speed(somSelf, ev, 0);
    }
```

Figure 6 (Part 2 of 2). Car — Updated C Template

Step 5: Compiling the Implementation Code

After updating the implementation templates, you can run the language compiler. Figure 7 shows JCL, which is extracted from the sample data set SOMMVS.SGOSJCL(GOS1AUTO), to C compile the code for the Vehicle and Car classes. In this example, the resulting object files are placed in temporary storage.

```
//CCCAR      EXEC EDCC,INFILE=&CDSN.(CAR),
//  OUTFILE='&&OBJ1,DISP=(NEW,PASS)',
//           CPARM='LO RENT DLL'
//COMPILE.SYSLIB DD
//               DD DSN=&HDSN.,DISP=SHR
//               DD DSN=&IDLPRFX..H,DISP=SHR
//               DD DSN=&SOM..SGOSH.H,DISP=SHR
//COMPILE.IH DD DSN=&IDLPRFX..IH,DISP=SHR
//CCVEHIC   EXEC EDCC,INFILE=&CDSN.(VEHICLE),
//  OUTFILE='&&OBJ2,DISP=(NEW,PASS)',
//           CPARM='LO RENT DLL'
//COMPILE.SYSLIB DD
//               DD DSN=&HDSN.,DISP=SHR
//               DD DSN=&IDLPRFX..H,DISP=SHR
//               DD DSN=&SOM..SGOSH.H,DISP=SHR
//COMPILE.IH DD DSN=&IDLPRFX..IH,DISP=SHR
```

Figure 7. JCL Statements to C Compile Vehicle and Car

Step 6: Prelinking and Linking the Object Code

The last step in building a new class is to prelink and link the object code to enable it for use by an application program. The prelinker produces an updated object deck that contains information to resolve the names and offsets of the methods and variables defined in the classes. The linkage editor then uses this object deck to create the DLL load module that contains the new classes. Figure 8 shows JCL statements, from the SOMMVS.SGOSJCL(GOS1AUTO) data set, to perform the prelinking and linking steps for Vehicle and Car.

```
//VEHICLE     EXEC EDCPL,LPARM='AMODE=31,MAP,RENT'
//PLKED.SYSDEFSD DD DSN=&&IMPORTS1(VEHICLE),DISP=(NEW,PASS),
//           UNIT=SYSDA,SPACE=(TRK,(3,3,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//PLKED.IMPORT   DD DSN=&SOM..SGOSIMP,DISP=SHR
//PLKED.SYSIN    DD  DSN=&&OBJ2,DISP=(SHR,DELETE)
//               DD  *
  INCLUDE IMPORT(GOSSOMK)
  NAME VEHICLE(R)
//LKED.SYSLMOD DD DSN=&&LOAD1(VEHICLE),DISP=(NEW,PASS),
//           UNIT=SYSDA,SPACE=(TRK,(3,3,1)),
//           DCB=(RECFM=U,LRECL=0,BLKSIZE=6144)
//*
//CAR         EXEC EDCPL,LPARM='AMODE=31,MAP,RENT'
//PLKED.SYSDEFSD DD DSN=&&IMPORTS2(CAR),DISP=(NEW,PASS),
//           UNIT=SYSDA,SPACE=(TRK,(3,3,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//PLKED.IMPORT   DD DSN=&SOM..SGOSIMP,DISP=SHR
//               DD DSN=&&IMPORTS1,DISP=(SHR,PASS)
//PLKED.SYSIN    DD  DSN=&&OBJ1,DISP=(SHR,DELETE)
//               DD  *
  INCLUDE IMPORT(GOSSOMK)
  INCLUDE IMPORT(VEHICLE)
  NAME CAR(R)
//LKED.SYSLMOD DD DSN=&&LOAD2(CAR),DISP=(NEW,PASS),
//           UNIT=SYSDA,SPACE=(TRK,(3,3,1)),
//           DCB=(RECFM=U,LRECL=0,BLKSIZE=6144)
```

Figure 8. JCL Steps to Prelink and Link Vehicle and Car Code

When you have created the object module containing the new classes, the next step is to create the application that will use these classes. For more information, see Chapter 4, “Using the Classes in an Application” on page 15.

Chapter 4. Using the Classes in an Application

Chapter 3, “Building the Classes” on page 7, took you through the steps of building new classes (Vehicle and Car) using OS/390 SOMobjects. This section demonstrates how to create an application program that uses these new classes. The tasks described in this section (writing, compiling, prelinking, and linking code) are accomplished by the samples provided with OS/390 SOMobjects.

Step 1: Determining What to Use

Just as you determine the characteristics of any new classes, you must determine what function your application program will perform. You may also need to consider the language in which the code will be written, how the code will be compiled, and how the code will be implemented.

This chapter shows the steps to create the “DRVCAR” application, which uses the Vehicle and Car classes to create an object called “BlueCar.” The code is written in C and JCL statements are used to compile, prelink, link, and run the code in batch mode.

Step 2: Creating the Application

Figure 9 shows the code for the DRVCAR application. This code is provided as a sample in the SOMMVS.SGOSSMPC.C(DRVCAR) data set. The DRVCAR code includes the implementation header file (`car.h`) that contains the prototypes for the methods that DRVCAR will use.

```
??=ifdef __COMPILER_VER__
   ??=pragma filetag ("IBM-1047")
??=endif

#pragma nosequence nomargins

/*
 * Application program that instantiates a Car object using the
 * SOM RTL dynamic linking and loading services.
*/

| #pragma runopts(POSIX(ON))

#include <stdio.h>
#include <car.h>

#define MAX_LEGAL_SPEED 55
#define MY_SPEED 40

int main( int argc, char *argv[]){

 int num_doors;
 Car BlueCar;                /* Declare a car object */
 Environment ev;             /* Declare the SOM environment */

 SOM_InitEnvironment(&ev);   /* Initialize the local SOM environment  */
```

Figure 9 (Part 1 of 2). DRVCAR Application Code

```
/* Create a new car */
BlueCar = CarNew();          /* Create object via somNew */

/* Set the doors, color, and speed by using accessor macros provided by SOM */
num_doors = atoi(argv[1]);                    /* Get the number of doors passed by the user */
Car__set_doors(BlueCar, &ev, num_doors);      /* Set the number of doors */
Car__set_color(BlueCar, &ev, "Blue");
Car__set_max_speed(BlueCar, &ev, MAX_LEGAL_SPEED);

/* The following code uses the get accessor methods to obtain information about the car        */
printf("This car has %i doors and is colored %s \n", Car__get_doors(BlueCar, &ev), Car__get_color(BlueCar, &ev) );
printf("The car's maximum speed is saved as %i \n", Car__get_max_speed(BlueCar, &ev) );

/* Start the car by using the Start method inherited from Vehicle class */
Car_start(BlueCar, &ev);

/* Execute the accelerate method defined on the Car class  */
Car_accelerate(BlueCar, &ev, MY_SPEED);   /* Accelerate to my chosen speed */

/* Check that maximum legal speed has been set in the Car class's max_speed instance variable */
printf("The car's maximum speed is now saved as %i \n", Car__get_max_speed(BlueCar, &ev) );
printf("The car's current speed is now saved as %i \n", Car__get_speed(BlueCar, &ev) );

Car_accelerate(BlueCar, &ev, 60);        /* Accelerate to my chosen speed */

/* Check that maximum legal speed has been set in the Car class's max_speed instance variable */
printf("The car's maximum speed is now saved as %i \n", Car__get_max_speed(BlueCar, &ev) );
printf("The car's current speed is now saved as %i \n", Car__get_speed(BlueCar, &ev) );

/* Execute the decelerate method defined on the Car class  */
Car_decelerate(BlueCar, &ev);

/* Stop the car  */
Car_stop(BlueCar, &ev);
printf("The car's maximum speed is now saved as %i \n", Car__get_max_speed(BlueCar, &ev) );
printf("The car's current speed is now saved as %i \n", Car__get_speed(BlueCar, &ev) );

/* Clean up storage and return  */
_somFree(BlueCar);            /* Free the local and remote object */
SOM_UninitEnvironment(&ev);   /* Uninitialize the local environment */

return(0);
}
```

Figure 9 (Part 2 of 2). DRVCAR Application Code

Step 3: Compiling the Application Code

The next step is to compile your application code. Figure 10 shows the JCL statements that enables the DRVCAR code to be compiled as a batch job. This JCL is extracted from the SOMMVS.SGOSJCL(GOS1AUTO) data set. The CPARM parameter identifies the compiler options used for this code

```
//DRVCAR     EXEC EDCCPLG,
//           INFILE=&CDSN.(DRVCAR),
//           CPARM='LO RENT DLL',
//           GPARM='/4'
//COMPILE.SYSLIB DD
//               DD DSN=&HDSN.,DISP=SHR
//               DD DSN=&IDLPRFX..H,DISP=SHR
//               DD DSN=&SOM..SGOSH.H,DISP=SHR
```

Figure 10. JCL Statements to Compile the DRVCAR Application

Step 4: Prelinking and Linking the Application

After you compile the DRVCAR code, the resulting object code must be prelinked with the Vehicle and Car class implementations and the SOM Kernel code. The code is then linked to create the application load module. Figure 11 shows the JCL procedures to prelink and link the DRVCAR code. This JCL is extracted from the SOMMVS.SGOSJCL(GOS1AUTO) data set.

```
//PLKED.SYSDEFSD DD DUMMY
//PLKED.IMPORT DD DSN=&&IMPORTS1,DISP=(SHR,DELETE)
//             DD DSN=&&IMPORTS2,DISP=(SHR,DELETE)
//             DD DSN=&SOM..SGOSIMP,DISP=SHR
//PLKED.SYSIN  DD
//             DD  *
  INCLUDE IMPORT(GOSSOMK)
  INCLUDE IMPORT(CAR)
  INCLUDE IMPORT(VEHICLE)
```

Figure 11. JCL Statements to Prelink and Link the DRVCAR Application

Step 5: Running the Application

After creating the application load module, the final step is to run the application itself. Figure 12 shows the JCL statements to run the DRVCAR application. This JCL is extracted from the SOMMVS.SGOSJCL(GOS1AUTO) data set.

```
//GO.STEPLIB   DD DSN=&LE.SCEERUN,DISP=SHR
//             DD DSN=&SOM..SGOSLOAD,DISP=SHR
//             DD DSN=&&LOAD1,DISP=(SHR,DELETE)
//             DD DSN=&&LOAD2,DISP=(SHR,DELETE)
//GO.SOMENV    DD DSN=&SOM..SGOSPROF(GOSENV),DISP=SHR
//
```

Figure 12. JCL Statements to Run the DRVCAR Application

Note: Before you can run any OS/390 SOMobjects application, you should check that OS/390 SOMobjects has been configured and is running on your system. Your system programmer would typically perform the configuration steps, which are described in the *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide*.

Reviewing the Output

Figure 13 shows excerpts from the output that is produced when you successfully run the DRVCAR application.

```
This car has 4 doors and is colored Blue
The car's maximum speed is saved as 55
The vehicle is started.  The engine is running idle
Accelerating to 40 MPH
Current speed = 0 MPH
Current speed = 1 MPH
Current speed = 2 MPH
.
.                                  (the car accelerates)
.
Current speed = 39 MPH
Current speed = 40 MPH
The car's maximum speed is now saved as 55
The car's current speed is now saved as 40
Requested speed was 60 MPH
Cannot exceed the maximum speed of 55 MPH
Accelerating to 55 MPH
Current speed = 40 MPH
.
.                                  (the car continues to accelerate)
.
Current speed = 54 MPH
Current speed = 55 MPH
The car's maximum speed is now saved as 55
The car's current speed is now saved as 55
Decelerating
Current speed = 55 MPH
Current speed = 54 MPH
 .
 .                                 (the car decelerates)
 .
Current speed = 2 MPH
Current speed = 1 MPH
Current speed = 0 MPH
The vehicle is stopped.  The engine is off
The car's maximum speed is now saved as 55
The car's current speed is now saved as 0
```

Figure 13. Output from the DRVCAR Application

Chapter 5. Running in a Distributed Environment

This section provides an overview of the process of building (or modifying) classes and client application programs for use in the distributed OS/390 SOMobjects environment. It also describes the process of registering the application server for use with OS/390 SOMobjects.

As described on page 4, Distributed SOM (DSOM) is one of the OS/390 SOMobjects frameworks. It allows applications, called client programs, to access objects and classes that are located remotely in other processes, systems, or computing platforms. The location of the object is transparent to the client program; that is, the client program accesses the object as if it were stored locally.

Creating Distributed Classes

Generally, when creating classes and applications for use in the distributed environment, you follow the basic steps that are described in Chapter 3, “Building the Classes” on page 7:

1. Determine the characteristics of the classes
2. Create or modify the source IDL for the classes
3. Run the OS/390 SOMobjects compiler
4. Update the implementation template for a specific language
5. Compile the implementation code for the chosen language
6. Prelink and link the resulting object code.

In many cases, the code you create for non-distributed applications can also be used to run in a distributed environment. You do not need to modify the IDL or re-compile any code.

However, in some situations, you may want to modify the source code for a class to take advantage of Distributed SOMobjects features. For example, the following sections take you through the steps of modifying and compiling the code for the Dvehicle and Dcar classes, which are similar to the Vehicle and Car classes shown in Chapter 3, “Building the Classes” on page 7.

Determining What to Use

When building classes in the distributed environment, you identify the attributes and methods that are needed to define a particular object. You can also identify the characteristics that can be inherited from another class.

The Dvehicle class defines two methods (start and stop) and three attributes (speed, doors, and color) and their accessor methods. Dcar also defines an attribute (max_speed), its accessor method, and two additional methods (accelerate and decelerate). Both classes are implemented in the C programming language.

Creating the IDL Code

As in the non-distributed environment, IDL code defines the basic structure and characteristics of the classes. This section provides an example of how to write (or modify) the IDL for a distributed class.

Creating the Dvehicle IDL Code

Figure 14 shows the IDL code for the Dvehicle class; this code is included in the SOMMVS.SGOSSMPI.IDL(DVEHICLE) data set. The IDL for Dvehicle is similar to the IDL shown in Figure 2 on page 8 for the Vehicle class. However, in a distributed environment, the process for handling data strings differs. As such, the IDL code for Dvehicle needs to reflect how this type of information is handled.

In the distributed environment, storage for data strings is allocated in both the client and the server applications. The override statements in Dvehicle allocate local storage for the color string (“Red”) in the Dvehicle application server (see page 25). The color parameter passed to the server on the call to the `dvehicle_set_color` method is copied to a local variable in the application server. The override process allocates and copies the color variable passed by the client application.

```
#include <somobj.idl>

interface Dvehicle : SOMObject
{
 /* Vehicle attributes */
 attribute short speed;
 attribute short doors;
 attribute string color;

 /* The Start function */
 void start();

 /* The Stop function  */
 void stop();

 implementation
 {
  color: noset;
  _get_color: object_owns_result;
  filestem = dvehicle;
  functionprefix = dvehicle;
  dllname = dvehicle;
  releaseorder: _get_speed, _set_speed,
         _get_color, _set_color,
         _get_doors, _set_doors,
         start, stop;
 };
};
```

Figure 14. Dvehicle — IDL Definition

Updating the Dcar IDL Code

The example in Figure 15 on page 21 shows the IDL code for a new class called “Dcar” (this IDL is provided in the SOMMVS.SGOSSMPI.IDL(DCAR) data set.) The IDL for this class is similar to the Car example shown in Figure 3 on page 9. In this case, only the references to the Dvehicle files require updates. Dcar does not require override statements because it does not handle any string data.

```
#include <Dvehicle.idl>
interface Car : Dvehicle
{
 /* Car attributes */
 attribute long max_speed;

 /* The Accelerate method */
 void accelerate(in long mSpeed);      /* Takes maximum speed from caller */

 /* The Decelerate function  */
 void decelerate();

 implementation
 {
  filestem = dcar;
  functionprefix = Dcar_;
  dllname = dcar;
  releaseorder: _set_max_speed, _get_max_speed,
         accelerate, decelerate;
 };
};
```

Figure 15. IDL Code to Define the Dcar Class

Running the OS/390 SOMobjects Compiler

After modifying the IDL for each class, you need to run the OS/390 SOMobjects compiler. The OS/390 SOMobjects compiler then produces an implementation template for each class. Figure 16 shows the JCL statements to run the OS/390 SOMobjects compiler for the Dvehicle and Dcar classes. This JCL is extracted from the SOMMVS.SGOSJCL(GOSSMPSC) sample data set.

```
// SET SOMLIB=SOMMVS
// SET SAMPLIB=IBMUSER.SAMPLES
// SET DSOMLIB=&SOMLIB..SGOSLOAD
// SET INDSN=&SOMLIB..SGOSSMPI.IDL
// SET SOMPRF=&SOMLIB..SGOSPROF(GOSENV)
// SET LIBPRFX=CEE
//SC       PROC M='',P=''
//SOMC     EXEC PGM=SC,REGION=1000M,
//  PARM=('&P. -d ''&SAMPLIB.'' ''&INDSN.(&M.)''')
//STEPLIB  DD DSN=&DSOMLIB.,DISP=SHR
//         DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SOMENV  DD DSN=&SOMPRF.,DISP=SHR
//SYSPRINT DD SYSOUT=*
//         PEND
//DCAR     EXEC SC,M=DCAR,P='-sh:ih'
//DVEHICLE EXEC SC,M=DVEHICLE,P='-sh:ih'
```

Figure 16. JCL Statements to Run the OS/390 SOMobjects Compiler

Updating the Implementation Template

The next step is to update the resulting implementation template as needed for each class. Figure 17 on page 22 shows the updated C template for Dvehicle; this code is provided in the SOMMVS.SGOSSMPC.C(DVEHICLE) data set. Line **6a** shows the start of the changes that were made to run the class in the distributed environment. Compare this example to the code defined for the Vehicle class (see page 12).

```
??=ifdef __COMPILER_VER__
   ??=pragma filetag ("IBM-1047")
??=endif

#pragma nosequence nomargins
#ifndef _MVS
   #define _MVS
#endif

#ifndef SOM_Module_dvehicle_Source
#define SOM_Module_dvehicle_Source
#endif
#define Dvehicle_Class_Source
#include <dd:ih(dvehicle)>

SOM_Scope void  SOMLINK dvehiclestart(Dvehicle somSelf,  Environment *ev)
{
    DvehicleData *somThis = DvehicleGetData(somSelf);
    DvehicleMethodDebug("Dvehicle","dvehiclestart");
    _speed = 0;
    printf("The vehicle is started.  The engine is running idle\n");
}

SOM_Scope void  SOMLINK dvehiclestop(Dvehicle somSelf,  Environment *ev)
{
    DvehicleData *somThis = DvehicleGetData(somSelf);
    DvehicleMethodDebug("Dvehicle","dvehiclestop");
    printf("The vehicle is stopped.  The engine is off\n");
}

/* For distributed objects, we must override the method to set the color */
6a SOM_Scope void  SOMLINK dvehicle_set_color(Dvehicle somSelf,  Environment *ev, string color)
{
    DvehicleData *somThis = DvehicleGetData(somSelf);
    DvehicleMethodDebug("Dvehicle","dvehiclestop");

    if (_color) SOMFree(_color);
    _color = (string) SOMMalloc(strlen(color) + 1);
    strcpy(_color, color);
}
```

Figure 17. Dvehicle — Updated C Template

Figure 18 shows the updated implementation template for the Dcar class, which is provided in the SOMMVS.SGOSSMPC.C(DCAR) data set. This code is similar to code for the Car class (see page 12).

```
??=ifdef __COMPILER_VER__
   ??=pragma filetag ("IBM-1047")
??=endif

#pragma nosequence nomargins
#ifndef _MVS
   #define _MVS
#endif

#ifndef SOM_Module_dcar_Source
#define SOM_Module_dcar_Source
#endif
#define Car_Class_Source
#include <dd:ih(dcar)>

SOM_Scope void  SOMLINK Dcar_accelerate(Car somSelf, Environment *ev, long mSpeed)
```

Figure 18 (Part 1 of 2). Dcar — Updated C Template

```
{
    int i;             /* Loop control variable */
    long targetspeed;  /* Target speed to accelerate to */
    CarData *somThis = CarGetData(somSelf);
    CarMethodDebug("Car","Caraccelerate");
    targetspeed = mSpeed;
    if (_max_speed < targetspeed)
     {
      printf("Requested speed was %i MPH \n", mSpeed) ;
      printf("Cannot exceed the maximum speed of %i MPH \n", _max_speed);
      targetspeed = _max_speed ;
     }
    printf("Accelerating to %i MPH \n", targetspeed);

    /* Show acceleration up to the desired speed */
    for (i = Dvehicle__get_speed(somSelf, ev); i <= targetspeed; i++)
     {
      printf("Current speed = %i MPH \n", i);
     }
   /* Save the current speed in the speed instance variable */
   Dvehicle__set_speed(somSelf, ev, targetspeed);
}

SOM_Scope void  SOMLINK Dcar_decelerate(Car somSelf,  Environment *ev)
{
    int i;             /* Loop control variable */
    CarData *somThis = CarGetData(somSelf);
    CarMethodDebug("Car","Cardecelerate");

    printf("Decelerating\n");
    for (i = Dvehicle__get_speed(somSelf, ev); i >= 0; i--)
     {
      printf("Current speed = %i MPH \n", i);
     }
   /* Save the current speed in the speed instance variable */
  Dvehicle__set_speed(somSelf, ev, 0);
}
```

Figure 18 (Part 2 of 2). Dcar — Updated C Template

Compiling the Implementation Code

The next step is to compile the C code for the Dvehicle and Dcar classes. As Figure 19 shows, this process is similar to compiling the Vehicle and Car classes (see page 13). This JCL is extracted from the SOMMVS.SGOSJCL(GOSSMPCC) data set.

```
// SET SOMLIB=SOMMVS
// SET SAMPLIB=IBMUSER.SAMPLES
// SET OUTPLKD=&SAMPLIB..OBJ
// SET OUTLIST=&SAMPLIB..LISTING
// SET INDSN=&SOMLIB..SGOSSMPC.C
// SET LIBPRFX=CEE
// SET LNGPRFX=CBC
//DCAR         EXEC EDCC,LPARM='AMODE=31,MAP,RENT',
//           INFILE=&INDSN.(DCAR),
//           LIBPRFX=&LIBPRFX.,
//           CPARM='DLL EXPO SO LO RENT EXP OPT(1) NOMAR NOSEQ SHOW'
```

Figure 19 (Part 1 of 2). JCL Statements to C Compile Dvehicle and Dcar

```
//COMPILE.SYSLIB DD
//               DD  DSN=&LIBPRFX..SCEEH.H,DISP=SHR
//               DD  DSN=&LIBPRFX..SCEEH.SYS.H,DISP=SHR
//               DD  DSN=&SAMPLIB..H,DISP=SHR
//               DD  DSN=&HSOMLIB..SGOS.H,DISP=SHR
//COMPILE.IH DD DSN=&SAMPLIB..IH,DISP=SHR
//SYSCPRT  DD  DSN=&OUTLIST.(DCAR),DISP=SHR
//DVEHICLE     EXEC EDCC,LPARM='AMODE=31,MAP,RENT',
//           INFILE=&INDSN.(DVEHICLE),
//           LIBPRFX=&LIBPRFX.,
//           CPARM='DLL EXPO SO LO RENT EXP OPT(1) NOMAR NOSEQ SHOW'
//COMPILE.SYSLIB DD
//               DD  DSN=&LIBPRFX..SCEEH.H,DISP=SHR
//               DD  DSN=&LIBPRFX..SCEEH.SYS.H,DISP=SHR
//               DD  DSN=&SAMPLIB..H,DISP=SHR
//               DD  DSN=&HSOMLIB..SGOS.H,DISP=SHR
//COMPILE.IH DD DSN=&SAMPLIB..IH,DISP=SHR
//SYSCPRT  DD  DSN=&OUTLIST.(DVEHICLE),DISP=SHR
```

Figure 19 (Part 2 of 2). JCL Statements to C Compile Dvehicle and Dcar

Prelinking and Linking the Object Code

After you compile the updated implementation templates, you can then prelink and link the resulting object code. Figure 20 shows JCL statements to perform the pre-linking and linking steps for Dvehicle and Dcar. This JCL is extracted from the SOMMVS.SGOSJCL(GOSSMPLK) data set. You may want to compare these statements to the JCL statements in Figure 8 on page 14 for the non-distributed classes.

```
// SET SOMLIB=SOMMVS
// SET SAMPLIB=IBMUSER.SAMPLES
// SET INOBJ=&SAMPLIB..OBJ
// SET OUTDSN=&SAMPLIB..LOAD
// SET SIDEDECK=&SAMPLIB..IMPORTS
// SET DSOMIMP=&SOMLIB..SGOSIMP
// SET LIBPRFX=CEE
//LINKER   PROC MEM=''
//PLKED   EXEC PGM=EDCPRLK,REGION=2048K,PARM='MAP NCAL'
//STEPLIB  DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSMSGS  DD  DSN=&LIBPRFX..SCEEMSGP(EDCPMSGE),DISP=SHR
//OBJ      DD  DSN=&INOBJ.,DISP=SHR
//IMPORT   DD  DSN=&SIDEDECK.,DISP=SHR
//         DD  DSN=&DSOMIMP.,DISP=SHR
//SYSLIB   DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=SYSDA,DISP=(NEW,PASS),
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DSN=&SIDEDECK.(&MEM.),DISP=SHR
```

Figure 20 (Part 1 of 2). JCL Steps to Prelink and Link Dvehicle and Dcar Code

```
//LKED   EXEC PGM=HEWL,PARM='MAP,RMODE=ANY,AMODE=31,RENT'
//SYSLIB   DD  DSN=&LIBPRFX..SCEELKED,DISP=SHR
//SYSLIN   DD  DSNAME=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//SYSLMOD  DD  DSNAME=&OUTDSN.(&MEM.),DISP=SHR
//SYSPRINT DD  SYSOUT=*
//         PEND
//DVEHICLE   EXEC LINKER,MEM='DVEHICLE'
//PLKED.SYSIN DD *
  INCLUDE OBJ(DVEHICLE)
  INCLUDE IMPORT(GOSSOMD)
  INCLUDE IMPORT(GOSSOMK)
  ALIAS DVEHICLE
  NAME DVEHICLE(R)
//DCAR      EXEC LINKER,MEM='DCAR'
//PLKED.SYSIN DD *
  INCLUDE OBJ(DCAR)
  INCLUDE IMPORT(DVEHICLE)
  INCLUDE IMPORT(GOSSOMD)
  INCLUDE IMPORT(GOSSOMK)
  ALIAS DCAR
  NAME DCAR(R)
```

Figure 20 (Part 2 of 2). JCL Steps to Prelink and Link Dvehicle and Dcar Code

Registering the Classes and Application Server

In a distributed environment, one or more application servers must be configured on your system and registered with OS/390 SOMobjects; the classes you will use must also be registered with OS/390 SOMobjects. An application server is the application that defines the implementation of one or more classes. A server application could be defined to support several related classes; for example, you can define one application server to support both the Dcar and Dvehicle classes. When it receives a request from a client application, OS/390 SOMobjects can then identify the application server that supports a particular class. This server application is then started and notified to create an instance of the class that was requested by the client application.

Several tasks, which are typically performed by a system programmer, are needed to configure and register the application server; however, each task must only be performed once. The application server must be defined to the OS/390 Workload Manager (WLM). As part of the WLM definition, the name of the load library that contains the class implementations must be specified; this load library is created when the classes are prelinked and linked. Security authorizations for the server must also be defined (using RACF, for example). In addition, the application server must be registered with the OS/390 SOMobjects subsystem; OS/390 SOMobjects provides the REGIMPL utility to perform this task. The *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* describes the configuration and registration steps in more detail.

Each class you create for use in a distributed environment must also be registered to OS/390 SOMobjects. The REGIMPL utility can also be used to perform this task. Before running any distributed application, you should contact your system programmer to ensure that the individual classes and the appropriate application server are registered to OS/390 SOMobjects.

Creating a Distributed Client Application

The basic steps of creating a distributed client application are similar to creating client applications in a non-distributed environment. As described in Chapter 4, “Using the Classes in an Application” on page 15, there are five steps to building a client application:

1. Determine what you need in your application
2. Create or update the code for the client application
3. Compile the client application code
4. Prelink and link the client application
5. Run the client application.

In the distributed environment, a few coding changes may be needed to ensure the client application will work correctly in the distributed environment. These changes are described in more detail in the following sections.

Updating the Client Application Code

Figure 21 shows the code for a client application called DRVCARD; this code is provided in the SOMMVS.SGOSSMPC.C(DRVCARD) data set. This application is similar to the DRVCAR application shown in Figure 9 on page 15; however, it has been modified for use in the Distributed SOMobjects environment.

```
   #define SOM_STRICT_IDL 1
   ??=ifdef __COMPILER_VER__
      ??=pragma filetag ("IBM-1047")
   ??=endif

   #pragma nosequence nomargins
|  #pragma runopts(POSIX(ON))

5a #include <somd.h>
   #include <dcar.h>

   #define MAX_LEGAL_SPEED 55
   #define MY_SPEED 40

   int main( int argc, char *argv[])
   {
    int num_doors;
    Car RedCar;                 /* Declare a Car object */
    Environment ev;             /* Declare the local SOM environment structure  */

    SOM_InitEnvironment(&ev);   /* Initialize the local SOM environment  */
5b  SOMD_Init(&ev);             /* Initialize DSOM */
    /* Create a new car in the Vehicle server  */
5c  RedCar = somdCreate(&ev, "Car", TRUE);       /* Create object in the Car server  */
    /* Set the doors, color, and speed using macros provided by SOM */
    num_doors = atoi(argv[1]);                        /* Get the number of doors passed by the user */
    Car__set_doors(RedCar, &ev, num_doors);           /* Set the number of doors */
    Car__set_color(RedCar, &ev, "Red");               /* Set the color           */
    Car__set_max_speed(RedCar, &ev, MAX_LEGAL_SPEED); /* Set the max. speed */
```

Figure 21 (Part 1 of 2). DRVCARD — Distributed Client Application Code

```
    /* The following code uses the get accessor methods to obtain information about the car        */
    printf("This car has %i doors and is colored %s \n", Car__get_doors(RedCar, &ev), Car__get_color(RedCar, &ev) );
    printf("The car's maximum speed is saved as %i \n", Car__get_max_speed(RedCar, &ev) );

    /* Start the car  */
    Car_start(RedCar, &ev);

    /* Execute the accelerate and decelerate methods on the target server  */
    Car_accelerate(RedCar, &ev, MY_SPEED);

    /* Check that current speed has been set in the Car class's speed instance variable */
    printf("The car's maximum speed is now saved as %i \n", Car__get_max_speed(RedCar, &ev) );
    printf("The car's current speed is now saved as %i \n", Car__get_speed(RedCar, &ev) );

    Car_accelerate(RedCar, &ev, 60);   /* Accelerate to my chosen speed */

    /* Check that maximum speed was not exceeded */
    printf("The car's maximum speed is now saved as %i \n", Car__get_max_speed(RedCar, &ev) );
    printf("The car's current speed is now saved as %i \n", Car__get_speed(RedCar, &ev) );

    /* Slow down and stop */
    Car_decelerate(RedCar, &ev);
    Car_stop(RedCar, &ev);

    /* Check that speeds  */
    printf("The car's maximum speed is now saved as %i \n", Car__get_max_speed(RedCar, &ev) );
    printf("The car's current speed is now saved as %i \n", Car__get_speed(RedCar, &ev) );

    /* Clean up storage and return  */
5d  _somdDestroyObject(SOMD_ObjectMgr, &ev, RedCar);
5e  SOMD_Uninit(&ev);             /* Shutdown DSOM */
    SOM_UninitEnvironment(&ev);   /* Uninitialize the local environment */
    return(0);
   }
```

Figure 21 (Part 2 of 2). DRVCARD — Distributed Client Application Code

When creating any application for the Distributed SOMobjects environment, you need to include the `somd.h` header file (**5a**). This file defines information (constants, global variables, and run-time interfaces) that is needed by Distributed SOM. The next statement (**5b**) is required to initialize the Distributed SOMobjects environment itself.

In the DRVCAR example, the CarNew() method was called to create the “BlueCar” object. However, because the DRVCARD application will run in a distributed environment, it must call the somdCreate() method (**5c**). In this case, the method call tells Distribute SOM to create a “RedCar” object wherever it can find an implementation of the Car class.

Before ending its processing, the client program must release the storage that was obtained for the objects that were created. Statement **5d** is specified in DRVCARD to release the storage acquired for the “RedCar” object. Finally, the last statement (**5e**) is required to uninitialize the Distributed SOMobjects environment that was established.

Compiling the Client Application

The next step is to compile your client application code. Figure 22 on page 28 shows JCL statements you can use to compile the DRVCARD code as a batch job. This JCL is extracted from the SOMMVS.SGOSJCL(GOSSMPCC) data set.

```
//DRVCARD  EXEC SOMCMP,MEM=DRVCARD
//DRVCARD      EXEC EDCC,LPARM='AMODE=31,MAP,RENT',
//           INFILE=&INDSN.(DRVCARD),
//           LIBPRFX=&LIBPRFX.,
//           CPARM='DLL EXPO SO LO RENT EXP OPT(1) NOMAR NOSEQ SHOW'
//COMPILE.SYSLIB DD
//               DD  DSN=&LIBPRFX..SCEEH.H,DISP=SHR
//               DD  DSN=&LIBPRFX..SCEEH.SYS.H,DISP=SHR
//               DD  DSN=&SAMPLIB..H,DISP=SHR
//               DD  DSN=&HSOMLIB..SGOS.H,DISP=SHR
//COMPILE.IH DD DSN=&SAMPLIB..IH,DISP=SHR
//SYSCPRT  DD  DSN=&OUTLIST.(DRVCARD),DISP=SHR
```

Figure 22. JCL Statements to Compile the DRVCARD Application

Prelinking and Linking the Client Application

After compiling the DRVCARD code, you must prelink the resulting object code with the Dvehicle and Dcar class implementations. You then link this code to create the application load module. Figure 23 shows JCL to prelink and link DRVCARD; the JCL is extracted from the SOMMVS.SGOSJCL(GOSSMPLK) data set.

```
//DRVCARD  EXEC LINKER,MEM='DRVCARD'
//PLKED.SYSIN DD *
  INCLUDE OBJ(DRVCARD)
  INCLUDE IMPORT(DCAR)
  INCLUDE IMPORT(GOSSOMTC)
  INCLUDE IMPORT(GOSSOMK)
  INCLUDE IMPORT(GOSSOMD)
  INCLUDE IMPORT(GOSNMNAM)
  ALIAS DRVCARD
  NAME DRVCARD(R)
```

Figure 23. JCL Statements to Prelink and Link the DRVCAR Application

Running the Application

After you create the application load module and ensure that OS/390 SOMobjects is configured correctly, you can run the application. Figure 24 shows JCL that you can use to run DRVCARD; OS/390 SOMobjects does not provide a sample data set for this task.

```
//GOSSMPSC JOB ',?','Programmer',MSGCLASS=H,NOTIFY=IBMUSER
//  SET SAMPLIB=IBMUSER.SAMPLES
//  SET OUTDSN=&SAMPLIB..LOAD
//DRVCARD  EXEC  PGM=DRVCARD,
//       PARM='POSIX(ON)/4'
//       REGION=40M,TIME=NOLIMIT
//STEPLIB   DD DSN=&OUTDSN,DISP=SHR
//          DD DSN=SOMMVS.SGOSLOAD,DISP=SHR
//          DD DISP=SHR,DSN=CEE.V1R7M0.SCEERUN
//SOMENV    DD DSN=SOMMVS.SGOSPROF(GOSENV),DISP=SHR
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSERR    DD SYSOUT=*
//CEESNAP   DD SYSOUT=*
```

Figure 24. JCL Statements to Run the DRVCARD Application

Chapter 6. For More Information

This section describes where you can find more information about the other services provided with OS/390 SOMobjects and advanced methods you can use. It also contains a summary of the examples shown in this book and the sample data sets provided with OS/390 SOMobjects.

OS/390 SOMobjects Library Reference

The following books in the OS/390 SOMobjects library describe the many basic and advanced features that are available with OS/390 SOMobjects.

Configuration and Administration Guide

The *OS/390 V2R4.0 SOMobjects Configuration and Administration Guide* describes how to configure and run OS/390 SOMobjects in distributed and non-distributed environments. This book is written for system programmers or for OS/390 SOMobjects administrators. It describes how to define OS/390 SOMobjects to the OS/390 Workload Manager (WLM), use the REGIMPL utility to register servers and classes in the implementation repository, and set up RACF security for OS/390 SOMobjects. It also provides the syntax of the OS/390 SOMobjects configuration file and environment variables for OS/390 SOMobjects.

This book provides step-by-step examples to demonstrate each configuration step. It also describes shows how you can use the sample data sets that contain the Vehicle, Car, and DRVCAR examples, which are described in this book, to verify the configuration steps on your system.

Programmer's Guide

The *OS/390 V2R4.0 SOMobjects Programmer's Guide* describes the features of object-oriented programming and OS/390 SOMobjects in more detail. Through additional examples, this book describes how you can use OS/390 SOMobjects to create classes and the applications that use them. It also contains detailed information about the OS/390 SOMobjects frameworks, including: OS/390 SOMobjects Kernel, Distributed SOMobjects, Interface Repository, Metaclass Framework, and the Emitter Framework.

Programmer's Reference, Volume 1

The *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 1* provides reference information about each class, method, function, and macro provided with the OS/390 SOMobjects Kernel and frameworks. This book is designed to help you use OS/390 SOMobjects to build object-oriented class libraries and to write application programs that use OS/390 SOMobjects class libraries or OS/390 SOMobjects frameworks.

Programmer's Reference, Volume 2

The *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 2* provides information about an implementation of the standard interfaces defined by the Object Management Group (OMG). It also explains the abstract interface definitions introduced by OS/390 SOMobjects Object Services, which are class libraries used to manage objects in distributed applications. This book details the relationship between standard interface definitions and OS/390 SOMobjects Object Services. This includes information about:

- Various approaches to implementing standards
- The OS/390 SOMobjects Object Services approach in providing implementations of the standards
- How the OS/390 SOMobjects Object Services implementations are documented.

Programmer's Reference, Volume 3

The *OS/390 V2R4.0 SOMobjects Programmer's Reference, Volume 3* provides reference information about each class and method that is a part of the OS/390 SOMobjects Collection classes.

Object Services

The *OS/390 V2R4.0 SOMobjects Object Services* book provides detailed information to help you use the OS/390 SOMobjects Object Services. Object Services provides the following set of service classes to help you manage many objects in a distributed environment:

- Externalization
- Object Identity
- Life Cycle
- Naming
- Object Services Server
- Persistent Object Service
- Security Service.

Messages, Codes, and Diagnosis

The *OS/390 V2R4.0 SOMobjects Messages, Codes, and Diagnosis* book is designed to help you diagnosis and debug any problems that may occur with OS/390 SOMobjects or your applications that use OS/390 SOMobjects classes and services. The following topics are presented:

- Preparing applications to use OS/390 SOMobjects debugging facilities and the Distributed SOMobjects trace facilities
- Analyzing and troubleshooting problems with the OS/390 SOMobjects run-time library
- Understanding error messages issued by the OS/390 SOMobjects frameworks
- Understanding error codes and messages associated with OS/390 SOMobjects subsystem abends
- Understanding error codes issued by the OS/390 SOMobjects frameworks.

Summary of Sample Data Sets

Table 4 summarizes the data sets that are supplied as samples with OS/390 SOMobjects and are used as examples throughout this book. These samples can also help you verify the configuration of OS/390 SOMobjects on your system. You can copy the samples and modify them to create your own class and applications.

Table 4. Summary of Sample Data Sets

Task	Where Shown	Sample Data Set Name
Building Classes in a Non-Distributed Environment		
Creating Vehicle IDL	Figure 2 on page 8	SOMMVS.SGOSSMPI.IDL(VEHICLE)
Creating Car IDL	Figure 3 on page 9	SOMMVS.SGOSSMPI.IDL(CAR)
Running the OS/390 SOMobjects compiler	Figure 4 on page 9	SOMMVS.SGOSJCL(GOS1AUTO)
Updating the C template for Vehicle	Figure 5 on page 12	SOMMVS.SGOSSMPC.C(VEHICLE)
Updating the C template for Car	Figure 6 on page 12	SOMMVS.SGOSSMPC.C(CAR)
C compiling the Vehicle and Car classes	Figure 7 on page 13	SOMMVS.SGOSJCL(GOS1AUTO)
Prelinking and linking the object code	Figure 8 on page 14	SOMMVS.SGOSJCL(GOS1AUTO)
Building Application Programs in a Non-Distributed Environment		
Creating the DRVCAR application	Figure 9 on page 15	SOMMVS.SGOSSMPC.C(DRVCAR)
C compiling the DRVCAR application	Figure 10 on page 16	SOMMVS.SGOSJCL(GOS1AUTO)
Prelinking and linking the object code	Figure 11 on page 17	SOMMVS.SGOSJCL(GOS1AUTO)
Running the DRVCAR application	Figure 12 on page 17	SOMMVS.SGOSJCL(GOS1AUTO)
Building Classes in a Distributed Environment		
Updating the Dvehicle IDL	Figure 14 on page 20	SOMMVS.SGOSSMPI.IDL(DVEHICLE)
Updating the Dcar IDL	Figure 15 on page 21	SOMMVS.SGOSSMPI.IDL(DCAR)
Running the OS/390 SOMobjects compiler	Figure 16 on page 21	SOMMVS.SGOSJCL.(GOSSMPSC)
Updating the C template for Dvehicle	Figure 17 on page 22	SOMMVS.SGOSSMPC.C(DVEHICLE)
Updating the C template for Dcar	Figure 18 on page 22	SOMMVS.SGOSSMPC.C(DCAR)
C compiling the Dvehicle and Dcar classes	Figure 19 on page 23	SOMMVS.SGOSJCL(GOSSMPCC)
Prelink and link the Dvehicle and Dcar classes	Figure 20 on page 24	SOMMVS.SGOSJCL(GOSSMPLK)
Building Client Applications in a Distributed Environment		
Creating the DRVCARD application	Figure 21 on page 26	SOMMVS.SGOSSMPC.C(DRVCARD)
Compiling the DRVCARD application	Figure 22 on page 28	SOMMVS.SGOSJCL(GOSSMPCC)
Prelink and link the DRVCARD application	Figure 23 on page 28	SOMMVS.SGOSJCL(GOSSMPLK)

Note: The following data sets also contain JCL to build other distributed examples that are provided with OS/390 SOMobjects; see the *OS/390 V2R4.0 SOMobjects Programmer's Guide* for more information.

- SOMMVS.SGOSJCL(GOSSMPSC)
- SOMMVS.SGOSJCL(GOSSMPCC)
- SOMMVS.SGOSJCL(GOSSMPLK)

GOS1AUTO Example

In Chapter 3, “Building the Classes” on page 7 and Chapter 4, “Using the Classes in an Application” on page 15, the steps to compile, link, and run the Vehicle and Car classes and the DRVCAR application are shown in excerpts from the SOMMVS.SGOSJCL(GOS1AUTO) sample data set. Figure 25 on page 32 shows the entire JCL from this data set.

```
//GOS1AUTO JOB <JOB CARD PARAMETERS >
//  SET SOM=SOMMVS
//  SET LE=CEE
//  SET CCXX=CBC
//  SET IDLPRFX=SOMMVS.SGOSSMPI
//  SET HDSN=SOMMVS.SGOSSMPH.H
//  SET CDSN=SOMMVS.SGOSSMPC.C
/*JOBPARM  T=1,L=50
//ORDER JCLLIB ORDER=(&CCXX..SCBCPRC,&LE..SCEEPROC)
//*--------------------------------------------------------------------
//* SC JCL procedure
//*--------------------------------------------------------------------
//SC       PROC INDSN=,
//         MEM=,
//         SCPARMS='',
//         SOMPRFX=&SOM.
//SOMC     EXEC PGM=SC,REGION=40M,
//   PARM=('&SCPARMS ''&INDSN.(&MEM.)''')
//STEPLIB  DD   DSN=&SOMPRFX..SGOSLOAD,DISP=SHR
//         DD   DSN=&LE..SCEERUN,DISP=SHR
//SOMENV   DD   DSN=&SOMPRFX..SGOSPROF(GOSENV),DISP=SHR
//SYSPRINT DD   SYSOUT=*
//        PEND
//*--------------------------------------------------------------------
//* SOM Compile the AUTO CLASS IDLs
//*--------------------------------------------------------------------
//SCCAR    EXEC SC,MEM=CAR,
//         INDSN=&IDLPRFX..IDL,
//         SCPARMS='-sh:ih'
//SCVEHIC  EXEC SC,MEM=VEHICLE,
//         INDSN=&IDLPRFX..IDL,
//         SCPARMS='-sh:ih'
//*--------------------------------------------------------------------
//* C Compile the class implementations
//*--------------------------------------------------------------------
//CCCAR      EXEC EDCC,INFILE=&CDSN.(CAR),
//  OUTFILE='&&OBJ1,DISP=(NEW,PASS)',
//           CPARM='LO RENT DLL'
//COMPILE.SYSLIB DD
//               DD DSN=&HDSN.,DISP=SHR
//               DD DSN=&IDLPRFX..H,DISP=SHR
//               DD DSN=&SOM..SGOSH.H,DISP=SHR
//COMPILE.IH DD DSN=&IDLPRFX..IH,DISP=SHR
//CCVEHIC   EXEC EDCC,INFILE=&CDSN.(VEHICLE),
//  OUTFILE='&&OBJ2,DISP=(NEW,PASS)',
//           CPARM='LO RENT DLL'
//COMPILE.SYSLIB DD
//               DD DSN=&HDSN.,DISP=SHR
//               DD DSN=&IDLPRFX..H,DISP=SHR
//               DD DSN=&SOM..SGOSH.H,DISP=SHR
//COMPILE.IH DD DSN=&IDLPRFX..IH,DISP=SHR
```

Figure 25 (Part 1 of 2). SOMMVS.SGOSJCL(GOS1AUTO) Sample Data Set

```
//*--------------------------------------------------------------------
//* Prelink the AUTO classes to create a DLL
//*--------------------------------------------------------------------
//VEHICLE     EXEC EDCPL,LPARM='AMODE=31,MAP,RENT'
//PLKED.SYSDEFSD DD DSN=&&IMPORTS1(VEHICLE),DISP=(NEW,PASS),
//           UNIT=SYSDA,SPACE=(TRK,(3,3,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//PLKED.IMPORT   DD DSN=&SOM..SGOSIMP,DISP=SHR
//PLKED.SYSIN    DD  DSN=&&OBJ2,DISP=(SHR,DELETE)
//               DD  *
  INCLUDE IMPORT(GOSSOMK)
  NAME VEHICLE(R)
/*
//LKED.SYSLMOD DD DSN=&&LOAD1(VEHICLE),DISP=(NEW,PASS),
//           UNIT=SYSDA,SPACE=(TRK,(3,3,1)),
//           DCB=(RECFM=U,LRECL=0,BLKSIZE=6144)
//CAR         EXEC EDCPL,LPARM='AMODE=31,MAP,RENT'
//PLKED.SYSDEFSD DD DSN=&&IMPORTS2(CAR),DISP=(NEW,PASS),
//           UNIT=SYSDA,SPACE=(TRK,(3,3,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//PLKED.IMPORT   DD DSN=&SOM..SGOSIMP,DISP=SHR
//               DD DSN=&&IMPORTS1,DISP=(SHR,PASS)
//PLKED.SYSIN    DD  DSN=&&OBJ1,DISP=(SHR,DELETE)
//               DD  *
  INCLUDE IMPORT(GOSSOMK)
  INCLUDE IMPORT(VEHICLE)
  NAME CAR(R)
//LKED.SYSLMOD DD DSN=&&LOAD2(CAR),DISP=(NEW,PASS),
//           UNIT=SYSDA,SPACE=(TRK,(3,3,1)),
//           DCB=(RECFM=U,LRECL=0,BLKSIZE=6144)
//*--------------------------------------------------------------------
//* C Compile the main program.
//*--------------------------------------------------------------------
//DRVCAR     EXEC EDCCPLG,
//           INFILE=&CDSN.(DRVCAR),
//           CPARM='LO RENT DLL',
//           GPARM='/4'
//COMPILE.SYSLIB DD
//               DD DSN=&HDSN.,DISP=SHR
//               DD DSN=&IDLPRFX..H,DISP=SHR
//               DD DSN=&SOM..SGOSH.H,DISP=SHR
//PLKED.SYSDEFSD DD DUMMY
//PLKED.IMPORT DD DSN=&&IMPORTS1,DISP=(SHR,DELETE)
//             DD DSN=&&IMPORTS2,DISP=(SHR,DELETE)
//             DD DSN=&SOM..SGOSIMP,DISP=SHR
//PLKED.SYSIN  DD
//             DD  *
  INCLUDE IMPORT(GOSSOMK)
  INCLUDE IMPORT(CAR)
  INCLUDE IMPORT(VEHICLE)
/*
//GO.STEPLIB   DD DSN=&LE.SCEERUN,DISP=SHR
//             DD DSN=&SOM..SGOSLOAD,DISP=SHR
//             DD DSN=&&LOAD1,DISP=(SHR,DELETE)
//             DD DSN=&&LOAD2,DISP=(SHR,DELETE)
//GO.SOMENV    DD DSN=&SOM..SGOSPROF(GOSENV),DISP=SHR
//
```

Figure 25 (Part 2 of 2). SOMMVS.SGOSJCL(GOS1AUTO) Sample Data Set

Communicating Your Comments to IBM

OS/390
SOMobjects
Getting Started

Publication No. GA22-7248-01

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
Getting Started

Publication No. GA22-7248-01

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
GA22-7248-01

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

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.

GA22-7248-01