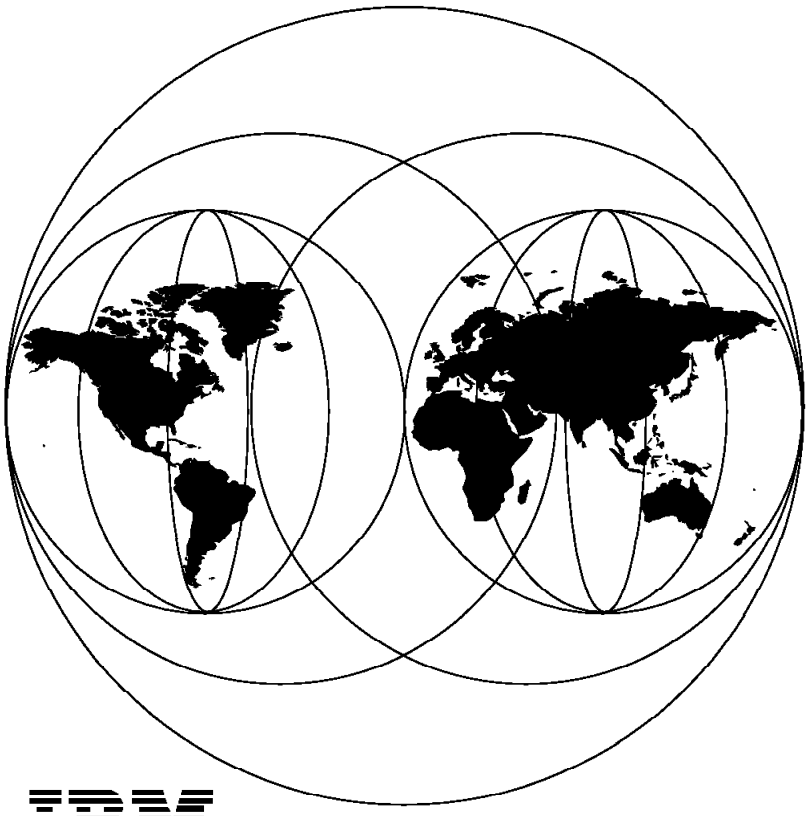
International Technical Support Organization

GG24-4473-00

Porting Applications to the OpenEdition MVS Platform

April 1995



IBM

International Technical Support Organization
Poughkeepsie Center

International Technical Support Organization

GG24-4473-00

Porting Applications to the OpenEdition MVS Platform

April 1995

Take Note!

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

First Edition (April 1995)

This edition applies to OpenEdition MVS, a feature of MVS/ESA System Product Version 5.1 (5655-068 and 5655-069).

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

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 541 Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

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

Abstract

This redbook describes our experiences with porting from UNIX to OpenEdition MVS, the runtime libraries and a sample application, generated from Sysdeco Innovation′s SYSTEMATOR 4GL/DATAMODELLER. Systemator is an advanced 4GL with a powerful data modelling capability, and a versatile, integrated environment for developing enterprise-wide client/server applications across a wide spectrum of platforms and relational databases.

This document was written for UNIX programmers to help guide them through sizing and managing a port to the OpenEdition MVS platform. The book describes the methodology used during the porting exercise. Some general porting hints and tips are included along with some specific considerations unique to the OpenEdition MVS platform.

Contents

Figures

Tables

Special Notices

This publication will help UNIX programmers port UNIX applications to MVS. The information in this publication is not intended as the specification of any programming interfaces that are provided by OpenEdition MVS. See the PUBLICATIONS section of the IBM Programming Announcement for OpenEdition MVS for more information about what publications are considered to be product documentation.

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 IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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

ACF/VTAM
C/370
DB2/6000
ES/9000
MVS/ESA
OS/2
S/390
AIX
DB2
DFSMS/MVS
IBM
OpenEdition
RS/6000

The following terms are trademarks of other companies:

Preface

This document will assist UNIX programmers in porting existing UNIX applications to the OpenEdition MVS platform. It will also assist managers and planners in determining the feasibility and effort involved in porting an application.

The document is primarily based on the experiences of one particular porting project carried out with IBM by a software vendor, but experiences of other ports have also been included.

The document is not intended to be a definitive guide on how to run a porting project. Such material is widely available elsewhere. This document concentrates on considerations unique to porting to OpenEdition MVS and on "hints and tips" acquired during this port and others.

A diskette accompanies this book and contains the source code or executables for the tools and examples discussed. The tools and code examples are provided on an "as is" basis and no formal support will be provided by IBM.

How This Document is Organized

The document is organized as follows:

- Chapter 1, "Summary"

  This chapter provides a Summary of our porting experience.

- Chapter 2, "Introduction"

  This chapter provides an introduction to the application that we ported to OpenEdition MVS.

- Chapter 3, "System Configuration"

  This chapter describes our porting system's configuration.

- Chapter 4, "Porting Methodology and Process"

  This chapter describes the methods that we used to port the application code to OpenEdition MVS.

- Chapter 5, "General Porting Experiences"

  This chapter provides a number of "hints and tips" that were derived from our porting experience.

- Chapter 6, "Platform Specific Considerations"

  This chapter describes porting considerations that are specific to the OpenEdition MVS environment.

- Chapter 7, "Migrating DB2 Programs"

  This chapter describes the process we used to port the source files that contained embedded SQL statements.

- Chapter 8, "Other Considerations for a Complete Port"

  This chapter describes a number of other considerations that need to be taken into account before, during and after porting to OpenEdition MVS.

- Appendix A, “Code Checker Utilities”

  This appendix provides a brief discussion on code checkers.

- Appendix B, “Uploading the Source Files from the Diskette”

  This appendix has the instructions on how to unload the sample diskette.

- Appendix C, “OpenEdition MVS Shell Command to Run TSO Commands”

  This appendix describes the `tso` shell command that can be used to execute certain TSO/E commands from the OpenEdition MVS shell.

- Appendix D, “Staying in Running Mode”

  This appendix provides some sample code that can be used to enable the shell to remain in RUNNING mode instead of dropping into INPUT mode.

- Appendix E, “Copying the C Header Files to the HFS”

  This appendix provides a REXX EXEC to copy the header files from MVS data sets into a directory in the HFS.

- Appendix F, “Modified Version of DB2 IVP Program for Testing CAF”

  This appendix provides sample C code that can be used to execute the DB2 installation verification program (IVP) from the OpenEdition MVS shell.

- Appendix G, “REXX EXEC to Run the DB2 Precompiler and BIND from the OpenEdition MVS Shell”

  This appendix provides a listing of the PRECOMP REXX EXEC that is used to run the DB2 precompiler from the OpenEdition MVS shell.

- Appendix H, “MVS Terminology”

  This appendix provides some common terms that are unique to the MVS environment.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document. Documents with an LY prefix in their order numbers are available to IBM-licensed customers only.

- *MVS/ESA Planning: OpenEdition MVS*, SC23-3015
- *MVS/ESA OpenEdition MVS User's Guide*, SC23-3013
- *MVS/ESA OpenEdition MVS Command Reference*, SC23-3014
- *MVS/ESA OpenEdition MVS Shell and Debugger Messages*, SC23-3780
- *MVS/ESA OpenEdition MVS Advanced Application Programming Tools*, SC23-3017
- *MVS/ESA Assembler Callable Services for OpenEdition MVS*, SC23-3020
- *Using REXX to Access OpenEdition MVS Services*, SC23-3803
- *OpenEdition MVS Supplement for rlogin and Associated Functions*, SC23-3847
- *IBM SAA AD/Cycle C/370 Library Reference*, SC09-1761
- *IBM SAA AD/Cycle C/370 Language Reference*, SC09-1762
- *IBM SAA AD/Cycle C/370 User's Guide*, SC09-1763

- *IBM SAA AD/Cycle C/370 Migration Guide*, SC09-1359
- *IBM SAA AD/Cycle C/370 Programming Guide for LE/370*, SC09-1840
- *IBM SAA AD/Cycle C/370 Library Reference: OpenEdition MVS Sockets*, SC23-3024
- *IBM SAA AD/Cycle Language Environment/370: Programming Guide*, SC26-4818
- *IBM SAA AD/Cycle Language Environment/370: Programming Reference*, SC26-3312
- *IBM SAA AD/Cycle Language Environment/370: Debugging Guide and Run-Time Messages*, SC26-4829
- *Portability Guide for IBM C*, SC09-1405
- *IBM DATABASE 2 Version 3: Administration Guide, Volume I*, SC26-4888
- *IBM DATABASE 2 Version 3: Application Programming and SQL Guide*, SC26-4889
- *IBM DATABASE 2 Version 3: Command and Utility Reference*, SC26-4891
- *IBM DATABASE 2 Version 3: Diagnosis Guide and Reference*, LY27-9603
- *IBM DATABASE 2 Version 3: Messages and Codes*, SC26-4892
- *IBM DATABASE 2 Version 3: Reference Summary*, SX26-3801
- *IBM DATABASE 2 Version 3: SQL Reference*, SC26-4890
- *Formal Register of Extensions and Differences in SQL*, SC26-3316
- *DATABASE 2 AIX/6000: SQL Reference*, SC09-1574
- *IBM TCP/IP for MVS: Customization and Administration Guide*, SC31-7134-00
- *IBM TCP/IP for MVS: Programmer's Reference*, SC31-7135
- *IBM TCP/IP for MVS: User's Guide*, SC31-7136
- *IBM TCP/IP for MVS: Performance Tuning Guide*, SC31-7188
- *IBM TCP/IP for MVS: Planning and Migration Guide*, SC31-7189
- *TCP/IP for OS/2: User's Guide*, SC31-6076
- *AIX Operating System: TCP/IP User's Guide*, SC23-2309
- *DFSMS/MVS Network File System User's Guide*, SC26-7028
- *DFSMS/MVS Network File System Customizaton and Operation*, SC26-7029
- *3390 Reference Summary*, GX26-4577
- *3380 Direct Access Storage Reference Summary*, GX26-1678
- *9340 DASD Subsystems Reference Summary*, GX26-3778
- *MVS/ESA JCL Reference*, GC28-1479
- *MVS/ESA JCL User's Guide*, GC28-1473

International Technical Support Organization Publications

- *MVS/ESA Support for IEEE POSIX Standards: Technical Presentation Guide*, GG24-3867
- *OpenEdition MVS for MVS/ESA 5.1, Presentation Guide*, GG24-4095

A complete list of International Technical Support Organization publications, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

To get a catalog of ITSO technical publications (known as “redbooks”), VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

How to Order ITSO Technical Publications

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their local IBM office.

Customers may order hardcopy ITSO books individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain books on a variety of products.

Acknowledgments

This project was designed and managed by:

Rich Conway
International Technical Support Organization, Poughkeepsie Center

The authors of this document are:

Kjell Arnfinn Andersen
IBM Norway

Trygve Brevik
Sysdeco Innovation AS

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Alan Smith
IBM UK

This publication is the result of a residency conducted at the International Technical Support Organization, Poughkeepsie Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Jan Baisden
International Technical Support Organization, Poughkeepsie Center

Roger Bonsteel
OpenEdition MVS Development

Rolf Hermanrud
IBM Norway

Susan LeVangia
OpenEdition MVS Information Development

Barry Lichtenstein
OpenEdition MVS Development

Maureen Moment
OpenEdition MVS Information Development

Jeff Noel
OpenEdition MVS Development

John Pfuntner
OpenEdition MVS Development

William Schoen
OpenEdition MVS Development

Rich Williams
OpenEdition MVS Development

Chapter 1. Summary

This book was written as the result of a project to port an application from UNIX to OpenEdition MVS.

The project was successfully completed on schedule with no major issues. The port was carried out by one experienced programmer from the application vendor with environmental support from IBM.

OpenEdition MVS was developed to link the UNIX world to the MVS mainframe world. When we started the project our goal was to provide a seamless environment where this port would be considered to the UNIX programmer, like any other UNIX port. In fact, the authors went to great lengths to provide tools so that the UNIX programmer didn't have to leave the UNIX world and venture over into anything that is remotely associated with traditional MVS. You will find those tools described in this book. Because of this work, the port was comparable to many other ports that the application vendor had previously made between many different UNIX and PC platforms.

The following are the main conclusions of the project:

- From an application programming viewpoint, OpenEdition MVS really does have the "look and feel" of UNIX.
- Porting applications to OpenEdition MVS is not significantly different from porting applications between UNIXes. Unique to OpenEdition MVS is the need to translate your ASCII data to EBCDIC.
- Access to some MVS skills is required during a port.

Chapter 2. Introduction

This book discusses a porting project carried out by Sysdeco Innovation AS on its Systemator product. The information given below is provided as background information and also to enable comparisons to be made between this situation and any potential porting project.

Sysdeco Innovation customers have a requirement to develop client/server applications that make existing mainframe data and functionality available to workstation based clients. Sysdeco Innovation undertook this project in order to meet that requirement. See *Introducing OpenEdition MVS* for further discussions on the relative strengths of the mainframe and workstation environments and the benefits of exploiting both through OpenEdition MVS.

2.1 Description of Systemator

Systemator is developed and marketed by Sysdeco Innovation AS. Sysdeco Innovation is a Norwegian company with an established customer base in Europe.

Systemator is a fourth generation development tool used for the specification, development and maintenance of database-oriented applications. Over twenty UNIX and PC environments are supported along with many database environments including DB2/MVS, DB2/2, DB2/6000, Oracle, Ingress, and Sybase.

Application development with Systemator is based on the use of data models. Systemator leads the application developer through the data modelling process using a graphical data model editor. Entities, data elements and relationships are defined together with a description of how the data will be displayed to the enduser.

Procedures to manipulate data described in the data model are written in Systemator's own programming language called Sysdul. The developer has the option to enhance these programs interactively.

The data model definitions and descriptions of procedures and screen pictures are held in Systemator's data dictionary together with all cross references between them. Any modification to the data model automatically leads to updating of all affected items in the data dictionary.

From the data dictionary, Systemator generates all application databases, application programs (in C) and documentation.

Systemator is run time and database platform independent. This is achieved by Systemator provided runtime libraries for each different environment. An application can be ported from one environment to another so long as the runtime libraries have been ported to the new environment.

All parts of the Systemator toolset are available on a set of UNIX systems, including AIX, HP-UX and SunOS. Applications using Systemator are developed and tested on these platforms. The application may then be generated by Systemator for use on other operating systems. Systemator applications can be moved to and executed on several UNIX systems (including OpenEdition MVS), and PC platforms (including DOS/Windows and OS/2 Presentation Manager).

The UNIX/PC based client software called SGUIRE (Systemator Graphical User Interface Run-time Environment) will handle all the user interaction. All the communication between the SGUIRE GUI interface and the application running on OpenEdition MVS will be done with stream sockets.

The DB2 database is modelled and the database scheme is generated with the Systemator toolset on the UNIX platform. The scheme is transferred to the MVS environment and the database is generated with the DB2 database generator.

Figure 1 shows an example of a user running a Systemator developed application with the server on an RS/6000 running AIX and DB2/6000 and the GUI client (SGUIRE) on a workstation.

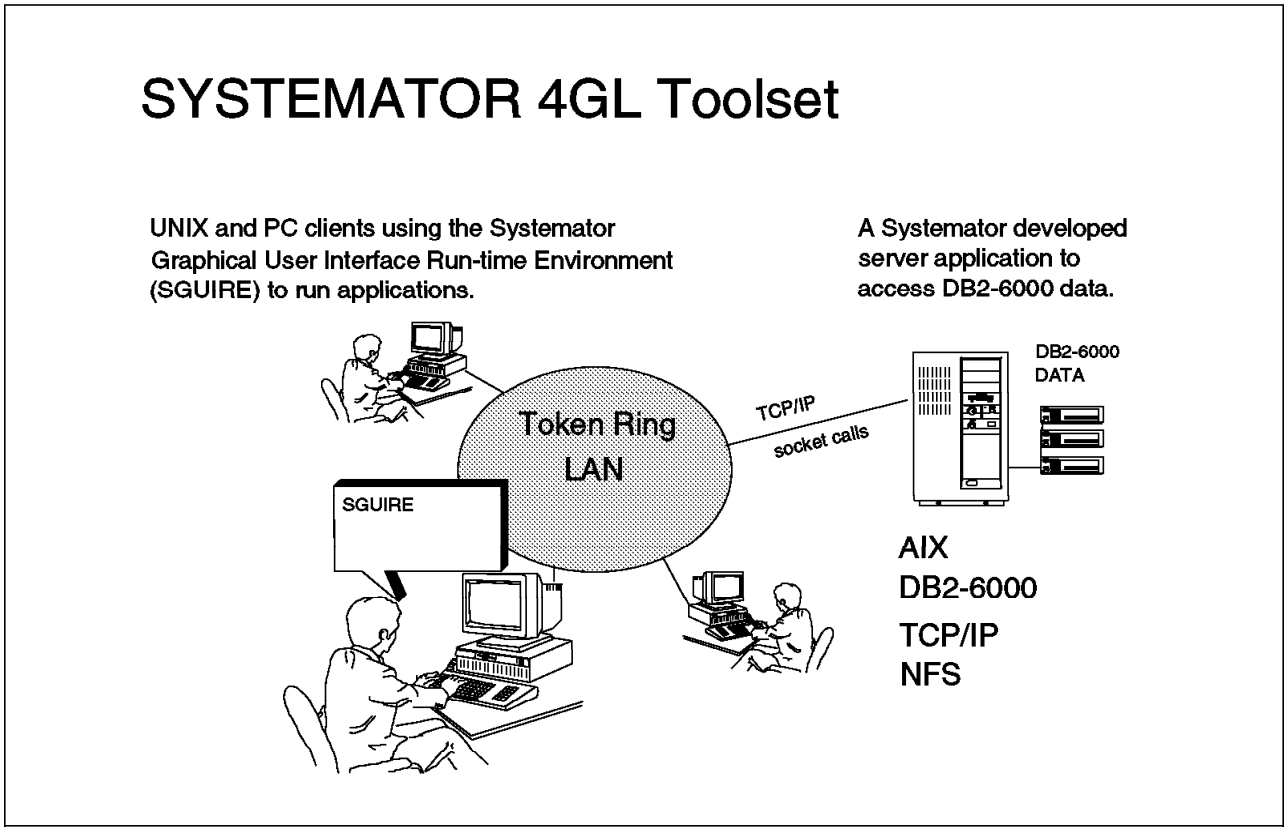


Figure 1. Structure of Typical Systemator Application Before this Porting Project

This same application was regenerated and the server piece of the application is now running under OpenEdition MVS, as shown in Figure 2 on page 5. The work involved in generating this application to run under OpenEdition MVS is no different than generating the same application to run on another UNIX platform.

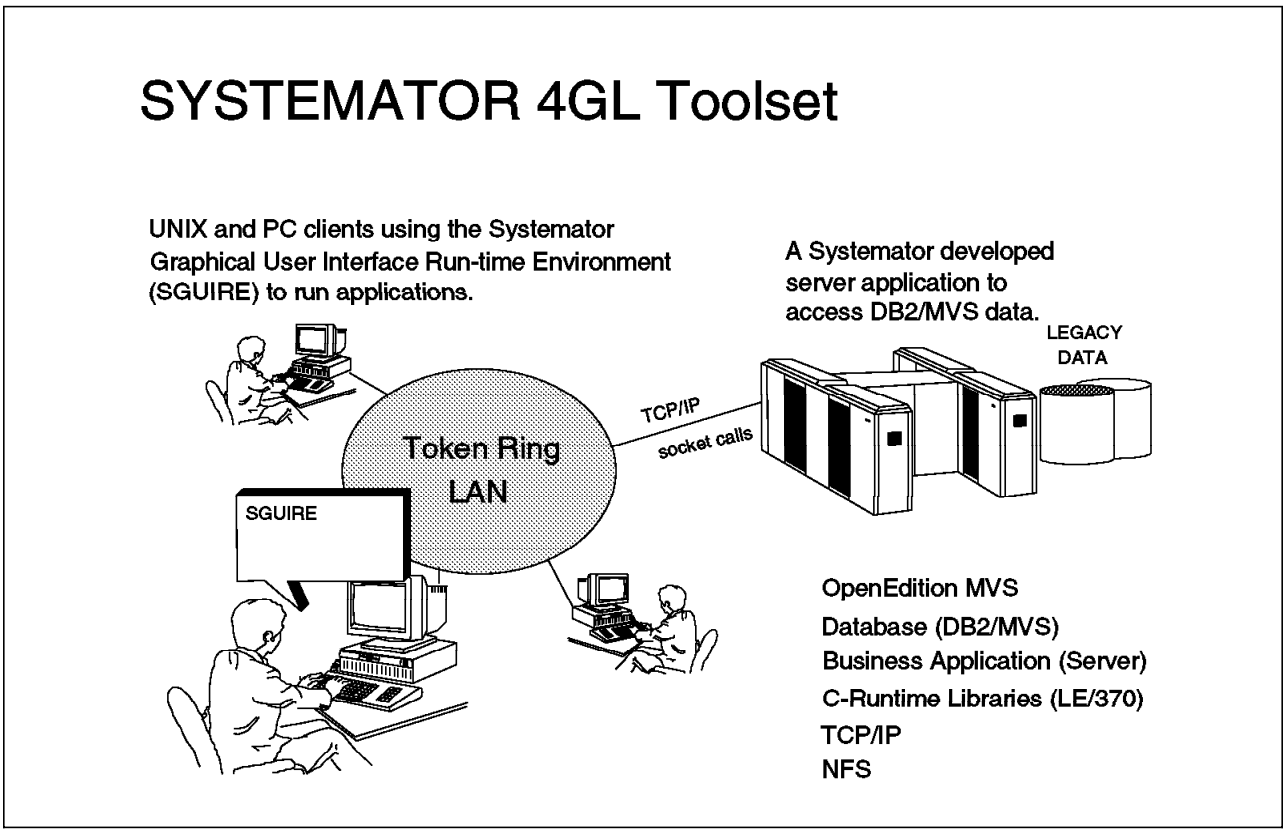


Figure 2. Structure of Typical Systemator Application After this Porting Project

2.2 Outline of Porting Project

The reason for the project was to enable Sysdeco Innovation customers to develop client/server applications in which the client is a workstation and the server is a mainframe.

The Systemator tool itself would continue to be run at a UNIX workstation, but the application generated would be a client/server application in which workstation based users would access data in a DB2/MVS database on an OpenEdition MVS server. There would be no application logic at the workstation. The application would be accessed through a GUI on UNIX or PC based workstations. There would be no 3270-style terminal attachment.

In order to achieve this, it was necessary to port the C runtime libraries to the OpenEdition MVS environment together with a business application. The application chosen, called CANTAS, is a sample data model used by Sysdeco Innovation in product documentation and on education courses. The model is based on a fictional company and includes the company's customers, agents, products, invoices and so on. Among the applications generated are order processing, agent commission and transportation. The sample is known to provide a good test of the port without any of the security or confidentiality implications of using actual customer applications.

In summary, it was neither the requirement nor the intent of the project to port the Systemator tool itself. The objectives were:

- To port the run-time libraries to allow OpenEdition MVS applications to be generated at a workstation
- To validate the port by using a sample application accessed from a workstation

2.3 Size and Complexity of this Port

The runtime libraries of Systemator comprise approximately 240,000 lines of C code in 850 files. The sample application comprises a data model with eight entity types, eight database tables, ten screen layouts and approximately 22,000 lines of C code.

The Sysdeco Innovation libraries have been ported many times between different hardware platforms and to provide support for different databases. The code has developed into highly portable code. The Sysdeco Innovation programmer conducting the port is very familiar with the source code and knew from experience which sections of code were likely to require changes. He is skilled in the porting process; this was his first port to the OpenEdition MVS platform.

Chapter 3. System Configuration

This chapter describes the system environment.

3.1 System Environment

The target system for the port was a LPAR on a 9121-570, which was configured with 64MB of central storage and 128MB of expanded storage with adequate DASD for system and user data. The RS/6000 workstation that was used by our UNIX programmer was a model 530H with 64MB of storage and 2GB of DASD storage for system and user source data. Table 1 contains the important software and levels that were running on the mainframe during the port. All the software that we ran was announced and generally available to IBM customers, except for a few locally written tools that are described later in this book.

Table 1. Important Mainframe Products and Levels That We Used

Product Name	**Program Number**	**Program FMID**	**VRM**
MVS/ESA 1. OpenEdition MVS System Services 2. OpenEdition MVS Application Services 3. OpenEdition MVS Shell and Utilities 4. OpenEdition MVS Debugger	5655-068 or 5655-069	HBB5510 1. HOM1120 2. HOT1120 3. HSU1120 4. HDX1120	5.1.0
AD/Cycle LE/370 RTL (Run Time Library)	5688-198	HMWL310	1.3.0
AD/Cycle C/370 Compiler	5688-216	HSQ4201	1.2.0
Data Facility System Managed Storage (DFSMS/MVS)	5695-DF1	HDZ11B0	1.2.0
Resource Access Control Facility (RACF)	5695-039	HRF2210	2.1.0
Transmission Control Protocol/Internet Protocol (TCP/IP)	5655-HAL	HTCP310	3.1.0
Interactive System Productivity Facility (ISPF)	5655-042	HPT4101	4.1.0
Network File System (NFS)	5695-DF1	HDZ11NP	1.2.0
DATABASE 2 (DB2)	5685-DB2	HDB3310 HIR1502 HIX3310 HIY3310 HIZ3310	3.0.0

Table 2 on page 8 contains the important software and levels that were running on the RS/6000 workstation during the port.

Table 2. Important RS/6000 Products and Levels That We Used

Product Name	Program Number
AIX Version 3.2.5 for RISC System/6000 • TCP/IP • NFS • X-Windows R5 • XL C Compiler 1.3	5756-030
DB2/6000 Version 1.1	5765-172

As shown in Figure 3, the LAN that contains the RS/6000 is connected to our production system, WTSCPOK, via a 3172 interconnection controller. Because our porting LPAR, C1JES2, already had a Systems Network Architecture (SNA) link to support our regular TSO/VTAM traffic through a 3088 channel to channel controller, we decided to use the same link and utilize the SNALINK LU0 interface of TCP/IP on MVS to provide our TCP/IP connectivity. This approach is fine for testing of function, but could raise performance issues if used in a production environment when sending many small packets across the line as with a client server application. A preferred configuration, shown in Figure 3 with a dotted line, would be to replace the use of SNALINK LU0 with a new hardware connection through another 3172 interconnection controller or connect the RS/6000 directly to the mainframe by installing a Block Multiplexer Channel Adapter card in the RS/6000. See *IBM TCP/IP for MVS: Customization and Administration Guide* and *IBM TCP/IP for MVS: Performance and Tuning Guide* for more information.

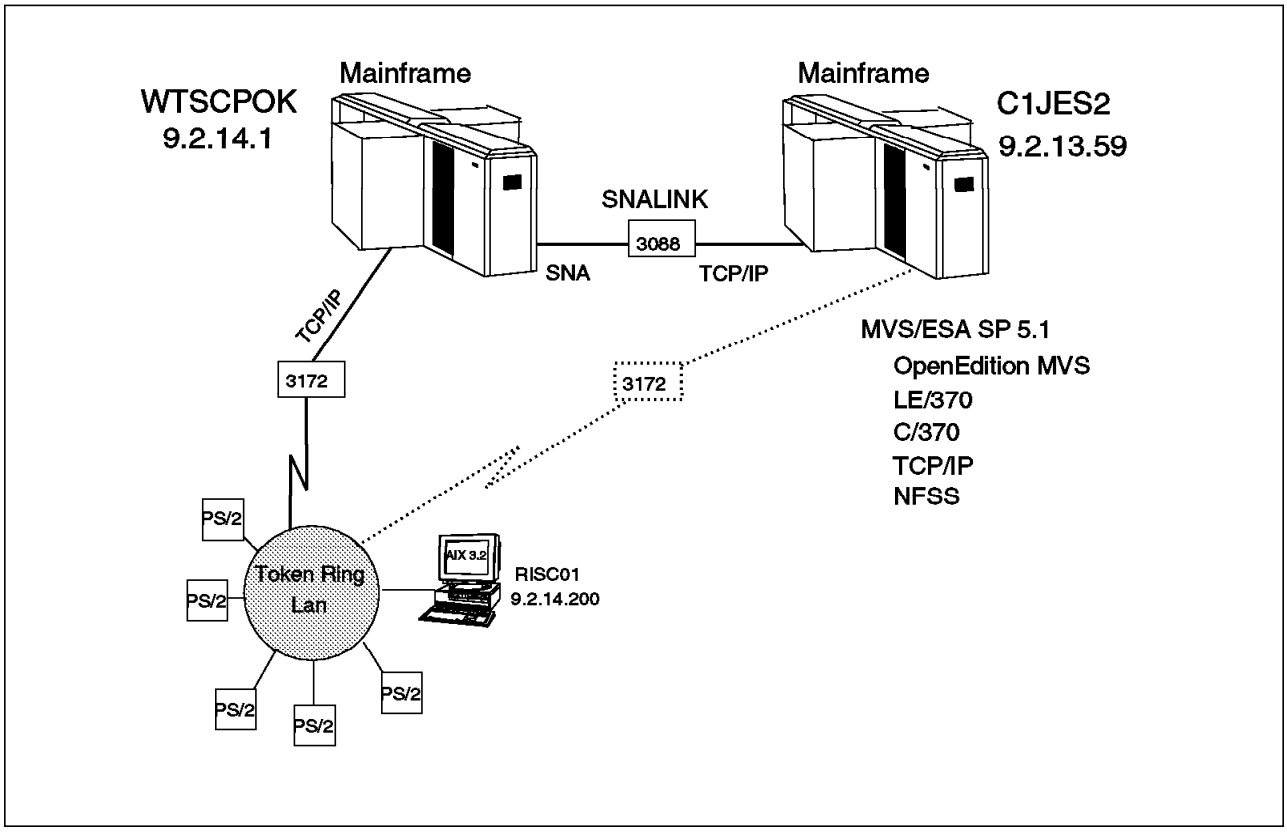


Figure 3. System Environment Configuration

Network File System (NFS) was used during the port. This gave us the ability to mount files in the hierarchical file system (HFS) to our RS/6000 workstation. By using this tool, editing of HFS files could be done on the workstation using the vi editor and the customer did not have to start learning how to use the ISPF editor, which is the common editor used in the MVS environment.

NOTE

See Chapter 8, “Other Considerations for a Complete Port” on page 67 for other considerations, including performance issues.

Chapter 4. Porting Methodology and Process

This chapter describes the methods that we used to port the Systemator toolset′s runtime libraries and a sample application to OpenEdition MVS.

As the standards groups, such as, IEEE POSIX, and X/Open work diligently to produce a set of standard operating interfaces and guidelines based on UNIX to write highly portable programs, in most cases, software that is ported doesn′t follow such guidelines. Even when good programming practices have been adhered to, this code contains a substantial amount of system specific code. Because of this, the porting process can be unstructured and a specific porting methodology is difficult to define. As more code is written to these standards, the porting process should become much easier.

Having the correct skills before starting a porting project is critical. C programming and debugging skills are essential for a successful port. Where modification of source code is necessary to get an application or tool running on another platform, it is greatly aided if the programmer doing the port is also familiar with the application and has experience porting this code to other platforms. In our case, the programmer was involved in many aspects of the design and coding of the Systemator toolset, so when a problem surfaced, he quickly focused in on the problem and was able to work out a resolution. In those cases where you are porting code that you are not familiar with, tools such as code checkers and lint filters may aid in identifying nonportable constructs in the code and also help you size the porting effort before you start. See Appendix A, “Code Checker Utilities” on page 71 for more information on code checker utilities.

4.1 Before You Start, Size the Port

It is useful in advance to attempt to size the port. How portable is the code? How much of it will need to be rewritten? How much effort is involved? To that end it is recommended that a list of all system calls made by the application be determined. This list can then be compared with the OpenEdition MVS supplied system calls. Any calls not available in OpenEdition MVS will have to be addressed during the port.

The method Sysdeco Innovation used was simply to attempt to link the applications without any libraries specified. Specifically, the linker *ld* was used on AIX to load the applications without the C library in the list of runtime libraries. This then resulted in unresolved or undefined reference errors for all system calls made. Figure 4 on page 12 shows the output from this command.

```
# ld signal
0706-317 ERROR: Unresolved or undefined symbols detected:
                Symbols in error (followed by references) are
                dumped to the load map.
                The -bloadmap:<filename> option will create a load map.
exit
printf
sscanf
errno
signal
```

Figure 4. linker (ld) Command to Find the List of System Calls

When compared with the list of function calls supported by OpenEdition MVS (using *IBM AD/Cycle C/370 Language Reference*), there were only two calls (out of a total of 113) missing:

- *crypt()*
- *putenv()*

crypt() was used within password checking routines. This part of the application was handled differently in OpenEdition MVS due to the differences in the security environment on MVS. See 6.3, “Coding and Running Daemons Under OpenEdition MVS” on page 46 for further details. Thus the absence of *crypt()* was not important for the port.

putenv() was used by a daemon to set an environment variable for Systemator′s own application debugger so it could be used from a variety of workstations and terminals. In OpenEdition MVS, the current environment variable must be set in advance. The application is now coded to issue a warning message if not set correctly.

It is IBM′s intention that both of these calls will be available in the next release of OpenEdition MVS, which is packaged with MVS/ESA V5.2.2.

4.2 The Porting Steps

The following steps will guide you in getting your source files installed on the MVS system. Our programmer brought the files that were needed for the port on 8mm tape cartridges created with the tar utility on his UNIX workstation.

4.2.1 Unloading the Source from Tape

Allocate sufficient space in the file system on the RS/6000 to hold the source code and restore the tapes. We used the tar utility to build the tapes, so we used tar to restore the tapes.

4.2.2 Get a HFS Data Set Defined On the OpenEdition MVS System

At this point you will need to ask your friendly MVS systems programmer to allocate some space in the OpenEdition hierarchical file system to store your source files.

OpenEdition MVS files are organized in a hierarchy, as in an AIX system. All files are members of a directory, and each directory is a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is

the root directory. OpenEdition MVS views an entire file hierarchy as a collection of hierarchical file system data sets (HFS data sets). A HFS data set is a new type of MVS data set for OpenEdition MVS. Each HFS data set is a mountable file system.

A file in the hierarchical file system is called a HFS file. A HFS data set can contain one or many HFS directories or files.

DFSMS/MVS facilities are used to manage a HFS data set.

4.2.2.1 MVS DASD Space Allocation

In the MVS world, disk space has traditionally been allocated in units of tracks or cylinders even if allocation in units of blocksize has been possible. The disadvantage of this method is the size of a track or cylinder often is different depending on the type of disk hardware. In more recent MVS versions it has been possible to allocate space in terms of megabytes (MB 1024*1024 = 1048576 bytes) or kilobytes (KB 1024 bytes). Programmers from the UNIX world are more familiar with this method of allocating space and can better estimate how much space should be allocated in the HFS to hold their source code.

Below is a short generalized table showing DASD allocation sizes for the most commonly used DASD families.

```
DASD type     Tracksize    Cylindersize  Trks/Cyl
-----------------------------------------------
3390          56KB         850KB         15
3380          47KB         712KB         15
9340          46KB         697KB         15
```

For more complete information about the different types of IBM direct access devices, see *3390 Reference Summary*, *3380 Direct Access Storage Reference Summary*, and *9340 DASD Subsystems Reference Summary*.

4.2.2.2 JCL for Allocating a HFS Data Set

The example below shows how your MVS systems programmer might allocate an OpenEdition MVS HFS data set with a size of 50 megabytes. The parameters used for allocation will vary depending on an installations local standards. A HFS data set must be a system managed data set.

```
//ALLOCHFS JOB (0,HFS),'HFS'
//ALLOC   EXEC PGM=IEFBR14
//HFS       DD DSN=OMVS.HFS,
//             SPACE=(1000,(50,1,1)),AVGREC=K,
//             DISP=(NEW,CATLG,DELETE),
//             DSNTYPE=HFS,
//             STORCLAS=STANDARD
//
```

This example will allocate a HFS data set with a primary space allocation of 1000*50*1024 (AVGREC=K) bytes, which gives a primary size of about 50MB and a secondary allocation of 1000*1*1024 (AVGREC=K) bytes, which gives a secondary allocation size of 1MB. A display of the data set is shown in Figure 5 on page 14.

```
Data Set Name . . . . . : OMVS.HFS

General Data                              Current Allocation
  Management class  . .  : STANDARD        Allocated kilobytes : 50,064
  Storage class . . . .  : STANDARD        Allocated extents . : 1
     Volume . . . . . .  : GNRL01          Maximum dir. blocks : NOLIMIT
     Device type  . . .  : 3390
  Data class  . . . . .  : PDS
     Organization . . .  : PO              Current Utilization
     Record format  . .  : U               Used pages  . . . . : 5
     Record length  . .  : 0               % Utilized  . . . . : 0
     Block size . . . .  : 0               Number of members . : 0
     1st extent kilobyte : 50064
     Secondary kilobytes : 1000
     Data set name type  : HFS
```

Figure 5. Display of the Attributes of the HFS Data Set

For further details regarding JCL syntax and use, refer to *MVS/ESA JCL Reference* and *MVS/ESA JCL User's Guide*. For further details on allocating HFS data sets, see *Planning: OpenEdition MVS*.

4.2.3 Using the OpenEdition MVS Shell

Before you can move your data files to OpenEdition MVS, you will need a user ID and password on the MVS system; one that contains an OMVS RACF segment. Have your MVS systems programmer set this up for you. See *MVS/ESA Planning: OpenEdition MVS* for more information.

Once you have a user ID and password you can access the OpenEdition MVS shell. There are currently two ways a user can access the OpenEdition shell:

- From a logged on TSO/E user ID using a dumb 3270 terminal or using a workstation running a 3270 emulator
- From a workstation running TCP/IP with the `rlogin` command

4.2.3.1 How to Login to the OpenEdition MVS Shell from TSO/E

In a System Network Architecture (SNA) network, remote users access TSO/E through Virtual Telecommunications Access Method (VTAM). In a TCP/IP network, remote users that have the Telnet 3270 client function access TSO/E by entering the TN3270 command. See the *AIX Operating System: TCP/IP User's Guide* or *TCP/IP for OS/2: User's Guide*, or documentation for your computer's 3270 emulation.

Figure 6 on page 15 shows a typical TSO/E logon panel where you enter your password.

```
------------------------------ TSO/E LOGON ----------------------------

   Enter LOGON parameters below:                  RACF LOGON parameter

   Userid    ===> KJELLAA

   Password  ===>                                 New Password ===>

   Procedure ===> KAAPROC                         Group Ident  ===>

   Acct Nmbr ===> MVS510

   Size      ===> 7500

   Perform   ===>

   Command   ===> ispf41

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice      S -Reconnect        -OIDcard
```

Figure 6. TSO/E Logon Panel

After you enter your password, you begin the logon process. Depending on how your TSO/E environment is setup by your MVS systems programmer, you can be placed in the native TSO/E environment, identified by the READY prompt, placed directly into the shell or placed into ISPF (Interactive System Productivity Facility), which is a set of front-end menus. Our system was setup to place each user into the ISPF menus as shown in Figure 7.

```
  Menu  Utilities  Compilers  Options  Status  Help
--------------------------------------------------------------------------
MVS/ESA 5.1               ISPF Primary Option Menu
Option ===> OS

0  Settings      Terminal and user parameters           User ID . : KJELLAA
1  View          Display source data or listings        Time. . . : 17:45
2  Edit          Create or change source data           Terminal. : 3278
3  Utilities     Perform utility functions              Screen. . : 1
4  Foreground    Interactive language processing        Language. : ENGLISH
5  Batch         Submit job for language processing     Appl ID . : ISR
6  Command       Enter TSO or Workstation commands      TSO logon : KAAPROC
7  Dialog Test   Perform dialog testing                 TSO prefix: KJELLAA
8  LM Facility   Library administrator functions        System ID : SY1
9  IBM Products  IBM program development products       MVS acct. : MVS510
10 SCLM          SW Configuration Library Manager       Release . : ISPF 4.1
OE OEDIT         Edit files in the HFS
OB OBROWSE       Browse files in the HFS
OI ISHELL        OpenEdition(TM) ISPF Shell
OS OMVS          OpenEdition(TM) Shell

Enter X to Terminate using log/list defaults
```

Figure 7. ISPF Menu Where the OpenEdition Shell is Selected

After the OpenEdition shell option is selected from the ISPF menu, the shell is invoked and displayed as shown in Figure 8 on page 16.

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/kjellaa: >




 ===>

ESC=# 1=Help    2=Subcmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
```

Figure 8. Invoking the Shell

4.2.3.2 Using rlogin from a Workstation to Access the Shell

If the inetd daemon is set up and active on the OpenEdition MVS system, you can use rlogin to access the shell from a workstation that has rlogin client support and is connected via TCP/IP to MVS. Figure 9 shows an example of issuing the rlogin command from AIX.

```
Last login: Mon Feb 23 10:06:09 EST 1995 on pts/0 from 9.12.14.117
# rlogin c1jes2 -l kjellaa
kjellaa's Password:

IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/kjellaa: >
```

Figure 9. rlogin Into the Shell from an AIX Machine

Once the login completes, you can run any program, including an X Windows application. For more information on using X Windows with OpenEdition MVS, see *TCP/IP for MVS Programmer's Reference*. You can rlogin into the shell more than once, using the same user ID and password; from TSO/E, you can login only once. However, you can be logged in to the shell both from TSO/E and using rlogin at the same time, using the same user ID and password.

There are a couple of limitations: you cannot use the ISPF editor during a rlogin session, and a rlogin session does not support Double Byte Character Set (DBCS) data. Ordinarily, you also would not be able to switch to TSO/E or run a TSO/E command from an rlogin session, but with the `tso` command that is supplied with this book, you have the ability to issue some TSO/E commands from an rlogin session. See Appendix C, "OpenEdition MVS Shell Command to Run TSO Commands" on page 77 for more information on the `tso` shell command. For more information on rlogin, see *OpenEdition MVS Supplement for rlogin and Associated Functions*.

4.2.4 Editors Available Under OpenEdition MVS

The traditional editor used in the OpenEdition MVS environment is an editor supplied as part of the ISPF product. In order to use this editor, you must be connected to a 3270 terminal or a workstation that is running a 3270 emulator and accessing the shell from a TSO/E session. You cannot use the ISPF editor during a rlogin session.

File editing is invoked using the `oedit` OpenEdition MVS shell command, while file browsing is invoked using the `obrowse` shell command.

The UNIX line editor called `ed` is also implemented in the OpenEdition MVS environment, but this is seldom used.

IBM intends to support the use of the `vi` editor from the shell in the next release of OpenEdition MVS which is packaged as part of MVS/ESA 5.2.2. Currently it is possible to use the `vi` editor only from your workstation against those files in the HFS if they are NFS mounted. See 4.2.5.1, "Using NFS to Transfer Files" for more information on using the `vi` editor after mounting a HFS directory to your workstation.

4.2.5 Moving Your Source Files to OpenEdition MVS

The source files were moved from AIX on the RS/6000 to the HFS data set on OpenEdition MVS. The OpenEdition MVS hierarchical file system is an EBCDIC based, byte oriented file system. See 6.1, "EBCDIC Character Encoding" on page 39 for information on what this means when copying data to OpenEdition MVS from ASCII workstations.

This movement of data could be done in several ways:

4.2.5.1 Using NFS to Transfer Files

If you have DFSMS/MVS NFS 1.2 installed on your OpenEdition MVS system and you have NFS client code installed on your workstation, NFS mounting your HFS directly to your workstation is the preferred way to copy data to, and access data from, the hierarchical file system

To access OpenEdition MVS files you must enter both the `mvslogin` command to log into the MVS host system and the `mount` command to mount the files to your

local system. Once the files are mounted to a local directory, you can read, write, create, delete and treat the mounted files as part of your workstation's local file system. ASCII-EBCDIC conversion for single-byte text files is performed automatically by means of default standard conversion tables. NFS does not provide conversion for double-byte text files. When you are finished with your work, use the umount and mvslogout commands to break the connection as shown in Figure 10. Using NFS to copy and access data from the workstation has shown to be the preferred method.

```
# mvslogin mvshost1 smith    1
Password required
GFSA973I Enter MVS password: password  2
GFSA955I smith logged in ok.
# mount mvshost1:"/hfs/smith,text,xlat(oemvs311)" /u/smith/mnt
                                                                  3
mount: mvshost1:"smith"
"smith" was attached successfully
# mount mvshost1:"/hfs/smith,binary" /u/smithbin/mnt  4

mount: mvshost1:"smith"
"smith" was attached successfully
# umount /u/smith/mnt     5
Unmounting '/u/smith/mnt' ... successful
# umount /u/smithbin/mnt     6
Unmounting '/u/smith/mnt' ... successful
# mvslogout mvshost1      7
UID 200 logged out ok
```

Figure 10. NFS Mount of a HFS Directory from an AIX Workstation

1 The TCP/IP hostname for the MVS system is mvshost1. The user ID you are using to login to the MVS system is smith.
2 Enter your password at the prompt.
3 Enter the mount command with the following:

mvshost1	TCP/IP hostname of the OpenEdition MVS system.
/hfs	HFS prefix: required by NFS to tell it that the directory to mount is a HFS directory and not a regular MVS data set.
/smith	HFS directory to be mounted.
text	Specifies that data be translated between ASCII and EBCDIC when updating the HFS file, and from EBCDIC to ASCII when the file is displayed at the workstation.
xlat(oemvs311)	Specifies the translation table named oemvs311 to be used to convert the data between ASCII (ISO 8859-1) and EBCDIC (1047 - OpenEdition MVS) when the data flows into the HFS and from EBCDIC (1047 - OpenEdition MVS) to ASCII (ISO 8859-1) when data is passed to the workstation.
/u/smith/mnt	Local mount point on the workstation.

Note: You can have the MVS systems programmer hardcode many of the attributes, such as text and xlat(oemvs311) in the NFS attributes data set and eliminate the need to code them in your client mount command, but these attributes will be in effect for all users of NFS on that MVS system. For more

information on the NFS attributes data set and to see what attributes need to be activated, see *DFSMS/MVS NFS Customization and Operation*.

4 Mount the same directory in the HFS, but mount it at a different mount point (/u/smithbin/mnt) and mount it as binary, so no translation takes place. This is useful if you want to copy binaries from the AIX workstation into the same HFS directory that you mounted in **3** above or run the od (octal dump) command, which requires that no translation takes place.

5 Unmount the HFS directory, /smith at mount point /u/smith/mnt.

6 Unmount the HFS directory, /smith at mount point /u/smithbin/mnt.

7 Logout from the MVS host.

NFS mount and copy was selected as the method for moving the files to the HFS data set. The use of NFS also made it possible to do most of the work on the workstation. The programmer could use the well-known vi editor instead of learning how to use the ISPF editor under OpenEdition MVS. After you have the HFS mounted, you can use AIX copy commands like the cp command, to copy your data from the workstation to the HFS.

Note: If you cannot use the mvslogin, mvslogout, or showattr commands, they might not be installed on your workstation. See *DFSMS/MVS NFS Customization and Operation* for information on how to install the mvslogin, mvslogout and showattr commands on the client workstation.

4.2.5.2 Use of TCP/IP FTP to Transfer Archive Files

If TCP/IP is installed on both the workstation and the MVS system, you can use the File Transfer Protocol (FTP) facility to transfer your source data. If transferring single-byte text data, FTP will convert the data from ASCII to EBCDIC for you. Binary data can also be sent using FTP if the binary parameter is specified.

Files are often bundled together in single files by utilities like pax, tar and cpio When these files are bundled into a single file, it is called an archive file. Usually the file name of the archive will indicate the utility that was used when the file was built. For example, a file named mvsport.tar indicates that the tar utility was used. The three utilities basically provide the same function: reading and writing of archive files. The important thing to know is tar and cpio can only read and write files of their respective formats, but pax can read or write in either format. So given a pax, tar, or cpio file, you can use pax from the OpenEdition MVS shell to explode or unwind the archive into its individual files.

To save disk space and transmission time, archive files can also be compressed with the tar, cpio and pax utilities by using the *-z* option. The naming convention that is generally used for a compressed archive file is to end the filename with a *.Z*, for example, mvsport.tar.Z.

Using FTP to transfer a file to the HFS is a two step process. You cannot transfer a file using FTP directly into the HFS. On the FTP PUT command, you must specify the name of a traditional MVS data set where you will temporarily load your archive file, then use the OPUT TSO/E command to copy the file into a directory in the HFS.

Note: We have found that it is best to preallocate a very large MVS data set to temporally hold your archive file before you invoke the FTP PUT command. When we let TCP/IP allocate the data set, it generally didn't allocate the data set large enough and the FTP PUT operation failed. Alternately, you can use the

FTP SITE command to pass the data set size attributes. See *TCP/IP User's Guide* for more information.

It is recommended with archive files to always use the BINARY transfer option. This applies when using the FTP GET and PUT commands as well as the TSO/E OPUT and OGET commands. If you don't use BINARY mode everywhere, your archive will most likely be corrupted and unusable.

To transfer a group of source files from an AIX workstation to MVS:

1. Archive the files you want to send with the `tar`, `cpio` or `pax` utilities.
2. Use FTP to send the archive to your MVS node using the binary parameter.
3. Use the OPUT command to copy it to the HFS using the binary parameter.
4. Use the pax utility with the translation parameters under OpenEdition MVS to explode or unwind the archive into its original files.

The example in Figure 11 shows the process of sending the C source code that resides in a directory on the RS/6000 called /u/smith/mvsport.

```
$ cd /u/smith
$ pwd
/u/smith
$ ls -l
-rwxr-----   1 smith    system       254 Jan  7 15:22 .profile
-rw-------   1 smith    system      2228 Jan  7 15:22 .sh_history
drwxr-xr-x   2 smith    system       512 Jan  7 15:22 mvsport
$ cd mvsport
$ pwd
/u/smith/mvsport
$ ls -l
-rw-r--r--   1 smith    system      3951 Jan  7 15:22 server.c
-rw-r--r--   1 smith    system      1395 Jan  7 15:22 passwd.c
-rw-r--r--   1 smith    system      4968 Jan  7 15:22 users.c
-rw-r--r--   1 smith    system      5147 Jan  7 15:22 sql.c
$ tar -cvf mvsport.tar /u/smith/mvsport
$ ftp c1jes2
Connected to c1jes2.itsc.pok.ibm.com.
220-FTPSERVE IBM MVS V3R1 at C1JES2.ITSC.POK.IBM.COM, 14:58:31 on 01/20/
220 Connection will close if idle for more than 5 minutes.
Name (c1jes2:root): wellie2
331 Send password please.
Password:
230 WELLIE2 is logged on.
ftp> binary
200 Representation type is IMAGE.
ftp> put /u/smith/mvsport/mvsport.tar 'wellie2.archive'
200 Port request OK.
125 Storing data set WELLIE2.ARCHIVE
250 Transfer completed successfully.
215040 bytes sent in 8.414 seconds (24.96 Kbytes/s)
ftp>
```

Figure 11. Copying Your Source Files to OpenEdition MVS Using FTP from AIX

The example in Figure 12 on page 21 shows the process of copying your archive file from the traditional MVS data set called WELLIE2.ARCHIVE, to a directory called /u/wellie2/mvsport that exists in the HFS using the OPUT TSO/E command. As you can see, we are using the locally written `tso` shell command, that is supplied with this book, to execute the OPUT command from the shell.

For more information on the tso shell command, see Appendix C, “OpenEdition MVS Shell Command to Run TSO Commands” on page 77.

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/wellie2: >
tso oput "'wellie2.archive' '/u/wellie2/mvsport/mvsport.tar' binary"
oput 'wellie2.archive' '/u/wellie2/mvsport/mvsport.tar' binary







 ===>

ESC=# 1=Help    2=Subcmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
```

Figure 12. Copying Your Source Files from an MVS Data Set to the HFS from the Shell

Alternatively, you can enter the OPUT command from option 6 of the ISPF menu, as shown in Figure 13.

```
  Menu  List  Mode  Functions  Utilities  Help
------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> OPUT 'WELLIE2.ARCHIVE' '/u/wellie2/mvsport/mvsport.tar' BINARY
```

Figure 13. Copying Your Source files from an MVS Data Set to the HFS from TSO/E

The pax command can read and write archives that are in cpio, tar or pax formats along with converting the data between the ISO8859-1 (ASCII) and IBM-1047 (EBCDIC) code pages. We used the pax utility from the shell to unwind and convert (from ASCII to EBCDIC) the archive file into its original files. This is shown in Figure 14 on page 22.

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/wellie2: >cd /u/wellie2/mvsport
/u/wellie2/mvsport: >




 ===> pax -o from=ISO8859-1,to=IBM-1047 -rf mvsport.tar

ESC=# 1=Help    2=Subcmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
```

Figure 14. Using the pax Utility to Unwind and Convert the Archive

4.2.6 Customizing the Shell

If this is your first time using the OpenEdition MVS shell, you might need to do some customization before you start to compile and link your code. Some things that might need to be customized are environment variables in your $HOME/.profile file and environment variables associated with the c89 utility. For more information, see *Planning: OpenEdition MVS* and the *OpenEdition MVS: User's Guide*.

4.2.6.1 AD/Cycle C/370 Compiler

The c89 utility is the interface to the C/370 compiler, the prelinker, and the linkage editor under OpenEdition MVS. The c89 utility can be invoked directly from the shell or from a batch job. See *AD/Cycle C/370 User's Guide* and *OpenEdition MVS Command Reference* for more information.

There were no differences between the C compiler on RS/6000 and the C compiler under OpenEdition MVS that required us to make code changes to the application we were porting.

4.2.6.2 Use of Make

Depending on the size of the application you are porting, a larger application will probably have a collection of makefiles that are used to build the object files and executables from the source files. The OpenEdition MVS shell has a make command that can be used in your makefiles to assist you in your porting work. The make command calls the c89 utility by default to compile and link programs specified in your makefiles. There are some differences in the way the make command works under OpenEdition MVS compared to other platforms. See 5.6, "Use of MAKE and the Compiler" on page 36 and *OpenEdition Advanced Application Programming Tools* for more information on the differences with the

make command. See *OpenEdition MVS Command Reference* for more information on the syntax of the make command.

4.2.6.3 OpenEdition MVS Header Files

There are a couple of environment variables that are used to point the c89 utility to the header files. Environmental variable _C89_INCDIRS is set, by default, to search the /usr/include directory. _C89_CLIB_PREFIX is set, by default, to search the MVS data set EDC.V1R2M0.SEDCDHDR, which includes the C/370 compiler header files. See *OpenEdition MVS Command Reference* for more information on the environmental variables that affect the c89 utility. To get access to the OpenEdition MVS socket header files when compiling, add the following options when you invoke the c89 utility:

-DMVS

-D_OPEN_SOCKETS

-I"//'SYS1.SFOMHDRS'"

For example, if you were compiling a program called server:

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/kjellaa: >




 ===> c89 -I"//'SYS1.SFOMHDRS'" -DMVS -D_OPEN_SOCKETS -o server server.c

ESC=# 1=Help    2=Subcmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
      7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
```

Note: You will need to check to see what high level qualifier your MVS systems programmer used for data set SFOMHDRS when OpenEdition MVS was installed. The -I option above should reflect the name of that data set on your local OpenEdition MVS system. See 5.5, "Header Files" on page 32 for more information on header files. Also see *AD/Cycle C/370 Release 2 Library Reference: OpenEdition MVS Sockets* for more information on OpenEdition MVS socket header files.

4.2.7 Compiling Source Code

Now that all the source code is loaded in the HFS, you have the c89 utility customized for your shell session, and you have taken a look at your makefiles and made any necessary changes, it is time to start compiling your code. As we said earlier, you can run your code through a code checker or lint filter that uses the OpenEdition MVS header files to possibly flag any nonconforming and unsupported constructs. Since our UNIX programmer was very experienced and

knew the code very well, he decided to start running his makefiles immediately. First, he compiled all the code, changing the makefiles to add the *-c* c89 option. After getting through all the compiles, he ran the archive command (ar) to create libraries; then he linked his program. This is the process we used when porting the runtime libraries and a small application to OpenEdition MVS.

4.2.7.1 Functions, Libraries and Header Files

To understand the compilation process, it helps to understand a bit about functions and where your program needs to look to find them.

All the Systemator code was written in the C language. For the most part, C programs are written using functions. Functions are sometimes referred to as subroutines, library functions, or just routines. You can write your own functions, but OpenEdition MVS provides a vast collection of function definitions that, in most cases, match a function definition that is required by the code you are porting.

Note: The next release of OpenEdition MVS to be packaged with MVS/ESA SP 5.2.2 will contain over 1100 XPG 4.2 system calls and commands, which is 800 new functions added to the 300 POSIX system calls and commands in OpenEdition MVS that is shipped with the now current release of MVS/ESA, MVS/ESA SP 5.1.

The function definitions are kept in libraries. In OpenEdition MVS the default directory set by the _C89_LIBDIRS environment variable is the /lib directory followed by the /usr/lib directory. This default is set to be consistent with other UNIX implementations, but the library functions in OpenEdition MVS are not contained in those directories. Instead, the MVS data sets that are installed with the LE/370 product, are used to resolve library functions. See the _C89_PLIB_PREFIX in the *OpenEdition MVS Command Reference* for more information on what LE/370 data sets are searched.

Functions used by a program should be declared in the program. This is true whether you create them or you use system calls provided by the platform. If you do not declare the functions, the C compiler will create default declarations that may or may not operate correctly. System calls do not need to be declared explicitly, as their declarations are part of the header and include libraries that you specify to gain access to the system calls. These libraries are generally installed in the directory /usr/include in files with the suffix of .h on most systems. In OpenEdition MVS the header files are installed in MVS data sets, so these data sets need to be searched to resolve any function declarations. How you get to these header files is discussed in 4.2.6.3, “OpenEdition MVS Header Files” on page 23 and 5.5, “Header Files” on page 32.

If you run with the c89 default settings, the following are true:

- The source filenames must end with a *.c* suffix.
- The *-c* option of the c89 command specifies that only compilations will be done.
- The archive filenames must end with a *.a* suffix.
- Unless you name the executable file with a *-o file*, c89 will name it a.out.

4.2.7.2 Conditional Compilation

One of the problems programmers have is writing code that can work on many different machines. In theory, C code is portable; in reality, many machines have little differences that must be accounted for. The compiler allows the programmer great flexibility in changing the way code is generated through the use of conditional compilation. In our case, the code we ported contained #ifdef and #endif statements, that had specific coding that exploited functions on different platforms like VMS and Solaris. When it was found that a function worked differently or a header file declaration was different under OpenEdition MVS, #ifdef and #endif statements were inserted in the code. For example:

```
#ifdef openmvs
#include <stdlib.h>
#else
#include <malloc.h>
#endif
```

A compiler switch, *-Dopenmvs*, allows openmvs to be defined on the c89 command invocation or in your makefile to include all the code in between the #ifdef openmvs and #endif statements. You should think about using this technique throughout your code for all the platforms on which your code runs.

Chapter 5. General Porting Experiences

This chapter describes a number of “hints and tips” acquired during the Systemator port to OpenEdition MVS. Also included are hints and tips derived from the experience of other ports carried out by IBM, software vendors and customers worldwide, which may be of general interest and applicability.

Many early experiences have resulted in fixes or small enhancements to OpenEdition MVS code. As these code changes are now generally available, there is no value in documenting them here. It is recommended that the latest maintenance level be used at all times.

Experiences that relate specifically to the way an application was written are not included.

5.1 Summary of Findings

Most porting exercises have found porting to OpenEdition MVS to be comparable to porting to any other UNIX. Some of the common conclusions include:

- Realization that every UNIX has its own enhancements beyond POSIX and its own idiosyncrasies.
- Use of the shell interface for 3270 is optional, but requires some getting used to.
- Access to MVS skills is required.
- Header files are different.
- Make has some differences.
- If you are running a client/server application using sockets, you will need to handle the ASCII to EBCDIC translation of all the data yourself.

5.2 Which Manual Do I Need?

A newcomer to MVS can find the large number of manuals available to be confusing. The following list is intended to make finding the right manual a little easier.

Table 3 (Page 1 of 2). Key Sections in the AD/Cycle C/370 Publications

Book Titles	Key Sections
AD/Cycle C/370 Language Reference SC09-1762	• Intro to C • Elements of C • Declarations and Definitions • Expressions and Operators • C Language Statements • Preprocessor Directives • Conforming to ANSI Standards • Conforming to POSIX 1003.1

Table 3 (Page 2 of 2). Key Sections in the AD/Cycle C/370 Publications

Book Titles	Key Sections
AD/Cycle C/370 Library Reference SC09-1761	• Header Files • Library Functions
AD/Cycle C/370 User's Guide SC09-1763	• Working With the Switchable Compiler • Compiler Options under OpenEdition MVS • LE/370 V3R1: Runtime Options • Compiling, Prelinking, Linking, and Running C/370 Programs under OpenEdition MVS • Utilities • C/370 Compiler and Prelinker Return Codes and Messages, EDC0nnn, EDC1nnn and EDC4nnn messages • BPXBATCH utility messages, BPXMnnnn messages • localdef, iconv, and genxlt Utility Messages (EDCnnnn messages) • IBM Supplied Catalog Procedures, CLISTs and EXECs
AD/Cycle C/370 Programming Guide SC09-1840	• Input and Output • Combining C and Assembler • Using Threads in an OpenEdition MVS Application • Reentrancy • Debugging C/370 Programs that use Input/Output • Handling Error Conditions and Signals • Optimizing Code • Interprocess Communication under OpenEdition MVS • Using Environment Variables • International Enabling and Locales • Code Set Conversion Utilities, genxlt and iconv • Using Built-In Functions
AD/Cycle C/370 Library Reference: OpenEdition MVS Sockets SC23-3024	• OpenEdition socket library functions • Socket Return Codes • FOMnnnnn messages

Table 4. Key Sections in the LE/370 Publications

Book Titles	Key Sections
AD/Cycle Language Environment/370: Programming Guide SC26-4818	• Linking, Loading, and Running under OpenEdition MVS Services • LE/370 and POSIX Signal Handling Interactions
AD/Cycle Language Environment/370: Programming Reference SC26-3312	• LE/370 Run-Time Options
AD/Cycle Language Environment/370: Debugging Guide and Run-Time Messages SC26-4829	• LE/370 Run-Time Messages, CEEnnnnn messages • C/370 Run-Time Messages, EDC5nnnn, EDC6nnnn, and EDC7nnnn messages • LE/370 Abend Codes, U4nnn abend codes • C/370 Abend and Reason Codes, 2100-2502, 4000

Table 5 (Page 1 of 2). Key Sections in the OpenEdition MVS Publications

Book Titles	Key Sections
OpenEdition MVS Shell and Debugger Messages SC23-3780	• FSUMnnnn messages • FDBXnnnn messages • /etc/init Exit Status Codes
Using REXX to Access OpenEdition MVS Services SC23-3803	• Using TSO REXX for OpenEdition MVS Processing • The Syscall Commands • REXX Return Codes • REXX Messages, BPXWnnnnn messages
OpenEdition MVS User's Guide SC23-3013	• Customizing the Shell • Working with Shell Commands • Writing Shell Scripts • Using OpenEdition MVS from MVS Batch, TSO/E and ISPF • Working with the Hierarchical File System • Editing Files • Copying Files • Transferring Files between Systems • Using awk
OpenEdition MVS Command Reference SC23-3014	• Shell Commands • TSO Commands Used to Work with the HFS • Running Shell Scripts Under MVS Environments

Table 5 (Page 2 of 2). Key Sections in the OpenEdition MVS Publications

Book Titles	Key Sections
OpenEdition MVS: Advanced Application Programming Tools SC23-3017	• Using OpenEdition lex and yacc • Using OpenEdition make • Debugging OpenEdition C/370 Programs
Application Development Reference: Assembler Callable Services for OpenEdition MVS SC23-3020	• Callable Service Descriptions • Callable Service Return Codes • Using Threads with Callable Services
MVS/ESA: System Messages, Volume 2 (ASB - ERB) GC28-1481	• OpenEdition Messages, BPXFnnnn, BPXInnnn, BPXMnnnn, BPXOnnnn, BPXPnnnn, BPXTnnnn messages
MVS/ESA: System Codes GC28-1486	• OpenEdition System Codes, EC_ completion codes
DFSMS/MVS Network File System User′s Guide SC26-7028	• Using OpenEdition MVS Files • Commands and Examples for Various Clients

5.3 C Portability Guidelines

Source code portability is the ability to take a program that can be compiled, linked and run on one platform with one compiler, and compile, link and run it on another platform.

The porting process is simplified if attention has been paid to writing portable C source code from the beginning. Many ports found that much of the effort in porting is attributable to the portability of the C source. C is a highly portable language; however, certain elements are highly platform dependent.

The *Portability Guide for IBM C* compares the IBM C products across the full IBM hardware range of personal computers, RS/6000 workstations, mid-range machines and mainframes. The book is aimed at those who wish to write portable code or who want to accomplish a port. The advice that it gives holds true for portability of C code between many platforms, whether IBM or non-IBM.

The following is a summary of the areas of difference between C products discussed in the *Portability Guide for IBM C*.

floating point representation: Different platforms use different methods to represent and manipulate floating-point numbers. A floating-point operation will not necessarily have identical results on all platforms.

character encoding: The encoding or internal hexadecimal representation of characters is not the same for all C products. MVS uses EBCDIC encoding, whereas most others use ASCII. However there are still differences between particular ASCII encodings and between particular EBCDIC encodings. Portable code should refer to the character itself rather than the encoding of the character.

Different character encodings produce different collating sequences used by sorts and compares. See 6.1.4, “Effect of ASCII/EBCDIC on Collating Sequence” on page 41 for further discussion.

data types: The number of bytes, bits or words for a given C data type varies among implementations of C languages. Portable code cannot depend on a C data type being a particular size. The *sizeof* operator should be used whenever the size of a type is required.

structures and unions: Each member in a structure is aligned along a particular address boundary depending on the type of the member. Different C languages use different alignments for different types to ensure that structures are used efficiently. Portable code cannot make any assumptions about the alignment of a member within a structure or about the size of the structure itself. *sizeof* and *offsetof* should be used instead.

Unions are sometimes used to map one type onto another or to extract some of the bytes from a construct. These uses of union are not portable because of character encoding and because bytes within words are not always stored in the same order. (Note that IBM C/370 is “Big-endian”). The shift operators should be used when it is required to extract bytes from a longer type.

prototyping: It is recommended that all declarations and definitions are of the prototype form. Otherwise different C compilers can produce different output as a result of different promotions.

pointers and casting: Portable code cannot depend on pointers being a particular size or having any portable encoding.

filenames: Methods and terminology used for naming files vary by platform. There is no single portable format for file names. The following approach should be used to make file names as portable as possible:

- Isolate the code that uses file names in separate compilation units
- Isolate the construction of the file name in an include file. Then all that is required when porting is either to change the file name in the include file or to include a different file.
- Use conditional compilation to determine which file-name format should be used for a particular file.

miscellaneous: There are many other sources of potential differences. These include whitespace, escape sequences, comments, keywords, constants, identifiers, type promotion, preprocessor directives, macro usage, reentrancy, and so on. Experience and a good understanding of the particular C environment will help here.

5.4 Portability of Shell Scripts

The following is a list of some of the more common shell commands found in other UNIXes that are not currently available in the OpenEdition MVS shell. This is not intended to be an exhaustive list. Many commands found in other shells, particularly the “systems administration” commands have no meaning in an MVS context and are not listed here. Porting a shell script with any of the following commands to OpenEdition MVS would currently require code changes.

at	dump	li	nice	who
cal	egrep	lpr/lpq	nm	
cron	file	man	pg	
del	fgrep	more	strings	

Notes:

1. We found the OHELP TSO/E command to be an acceptable substitute for the `man` command.
2. The functions of the `at` and `cron` commands can be provided by a batch scheduling product such as OPC/ESA.
3. The function of `li` is provided by `ls` with a different syntax and output format. This could be significant for a shell script using the output of `li` as input to another command.
4. The `nm` command would be useful during porting to locate undefined symbols. We used the OpenEdition MVS documentation instead in order to determine symbol information.
5. We found the absence of `pg` and `more` to be a usability issue. One alternative is to use OEDIT. IBM intends to provide these commands in the next release of OpenEdition MVS.
6. IBM intends to provide `at`, `cal`, `cron`, `egrep`, `file`, `fgrep`, `man`, `more`, `nice`, `pg`, `strings`, and `who` in the next release of OpenEdition MVS that is packaged with MVS/ESA V5.2.2.

5.5 Header Files

When comparing AD/Cycle C/370 with the many other C compilers, it is inevitable that there will be some differences in the content of some header files and the syntax and availability of some functions.

Many additional functions have been added to AD/Cycle C/370 to provide the POSIX APIs for OpenEdition MVS. At the same time, compatibility has been maintained for traditional MVS applications. Each traditional UNIX has additional (and differing) functions beyond the POSIX standards. One of the difficulties that standards bodies have is that the many different UNIXes in use are not identical.

Many of these additional functions found in traditional UNIXes are to be added in later releases of OpenEdition MVS.

Header files on OpenEdition MVS are stored in MVS data sets. There is no direct access to these data sets from within the shell. That is, it is not possible to use shell commands, such as grep, to search header files. When investigating where a particular function is implemented in C/370, you can use the documentation. *AD/Cycle C/370 Library Reference* is particularly useful. Alternately, you could use TCP/IP FTP to copy the header files into a directory on your workstation or you can copy them into a directory in the HFS. A REXX EXEC called HDRCPY is supplied in Appendix E, “Copying the C Header Files to the HFS” on page 81 and can be used to make a copy of the header files into a directory of your choice in the HFS. After running HDRCPY, you can use grep or other utilities to search through this copy of the header files.

Note: You should continue to have c89 point to the header files that reside in MVS data sets to pick up the latest maintenance.

5.5.1 Differences with Header Files

Missing header files are usually indicative of a larger porting issue involving unsupported services. An application that requires a header file that is not available on OpenEdition MVS is probably trying to call a system service that is not supported. For example, OpenEdition MVS does not currently support streams and therefore the header file *stropts.h* is not to be found on OpenEdition MVS.

Header files on different systems contain different sets of system calls. Some header files on OpenEdition MVS include system calls that would be in multiple headers on other systems. Therefore, a header specified in a C program written for Solaris might not be required for correct compilation on OpenEdition MVS.

A simple way to determine the system calls and variable types required is to comment out the include statements and see what the compiler flags as unknown. This method also overcomes the situation where apparently missing header files are not actually required. This stems from the programming practice of including header files that aren't really needed by the program. That is, the function is not actually "missing." Another approach often adopted here is to define dummy or empty header files in the hope that the function actually required is provided elsewhere.

A general circumvention to allow successful compilation (but not, of course, to provide the missing function) is to copy the missing files from a platform where they do exist to OpenEdition MVS into a separate directory that is then passed to the compiler with the -I option.

The following are examples of include files that exist on some other UNIX platforms but which are either missing on OpenEdition MVS or for which there are differences in implementation. Where possible, advice is given on potential replacements based on the requirements of particular ports. The use of alternative functions should be fully researched to ensure their appropriateness.

curses.h: See the discussion in 6.2, "Interactive Terminal Connection" on page 43 to understand the options available now and in the future for *curses* function on OpenEdition MVS.

errno.h: In OpenEdition MVS, *errno* is a #define (macro) as opposed to an external integer variable in other UNIXes. This could be a problem when porting code that sets this variable.

The reason for the OpenEdition MVS implementation is that there were two conflicting requirements to have an extern symbol errno, and to have separate errno's for each thread.

The macro expansion should work correctly, but if the external symbol must be retained, then the circumvention is simply to undefine errno:

```
c89 -Uerrno ...
```

but if #include <errno.h> is also used then

```
#undef errno
```

must be used (instead) after that.

Note: It is IBM′s intention to make errno.h available in the next release of OpenEdition MVS that will be packaged with MVS/ESA V5.2.2.

lockf.h: is not found in OpenEdition MVS. Consider using fcntl.h for function fcntl().

manifest.h: Ensure that manifest.h is included from SFOMHDRS (the OpenEdition MVS socket header library) rather than from SEZACMAC (the TCP/IP socket header library) if using OpenEdition MVS socket functions.

malloc.h/memory.h: are not found in OpenEdition MVS. Consider using string.h or stdlib.h instead for function prototypes for *malloc(), memcmp(), memcpy(),...*

poll.h: is not found in OpenEdition MVS. I/O multiplexing is not yet part of POSIX.1. The poll function is part of UNIX System V, Release 4.

Note: It is IBM′s intention to make poll.h available in the next release of OpenEdition MVS that will be packaged with MVS/ESA V5.2.2.

select.h: is not found in OpenEdition MVS. I/O multiplexing is not yet part of POSIX.1. The select function is part of UNIX System V, Release 4 and 4.3+Berkeley Software Distribution.

stdio.h: The file structure does not follow the structure of other Unixes. The following usage is not accepted:

```
#ifdef NON_UNIX_STDIO
 {extern char *malloc(); setbuf(stderr, malloc(BUFSIZ));}
#else
 stderr->_flag &= _IONBF;
#endif

#define buf_end(x) (x->_flag & _IONBF ? x->_ptr : x->_base + BUFSIZ)
```

syslog.h: is not found in OpenEdition MVS.

Note: It is IBM′s intention to make syslog.h available in the next release of OpenEdition MVS that will be packaged with MVS/ESA V5.2.2.

sys/sysmacros.h: is not found in OpenEdition MVS. Consider using stat.h to find major and minor device numbers.

sys/file.h: is not found in OpenEdition MVS.

Note: It is IBM′s intention to make sys/file.h available in the next release of OpenEdition MVS that will be packaged with MVS/ESA V5.2.2.

sys/ioctl.h: is not found in OpenEdition MVS. Consider using termios.h instead.

Note: It is IBM′s intention to make sys/ioctl.h available in the next release of OpenEdition MVS that will be packaged with MVS/ESA V5.2.2.

sys/ipc.h, sys/sem.h, sys/shm.h: OpenEdition MVS does not currently support shared memory or semaphore methods of interprocess communication. Applications that use these will need to be recoded. OpenEdition MVS does implement the concept of a saved user ID, allowing the effective user ID to be switched back and forth between the real user ID and the superuser ID. This approach was used in one port.

Note: It is IBM′s intention to make sys/ipc.h, sys/sem.h, and sys/shm.h available in the next release of OpenEdition MVS that will be packaged with MVS/ESA V5.2.2.

sys/mode.h: is not found in OpenEdition MVS. Consider using modes.h instead.

sys/param.h: is not found in OpenEdition MVS. In particular,

```
#define MAXHOSTNAMELEN
```

is missing.

sys/stropts: OpenEdition MVS does not currently support streams.

sys/termio.h: is not found in OpenEdition MVS. Consider using termios.h instead.

types.h: The following type definitions are missing from types.h on OpenEdition MVS.

```
typedef  long   key_t;
typedef  long   vmhandle_t;
```

Also ensure that include picks up *types.h* from SEDCDHDR rather than from SEZACMAC which is, in fact, *rpc/types.h*.

5.5.2 Missing Function Calls

The following functions are not supported in this release of OpenEdition MVS. It is IBM′s intention to make the following functions available in the next release of OpenEdition MVS that is packaged as part of MVS/ESA V5.2.2:

flock()

index(): Use strchr() instead

initgroups()

ioctl()

mktemp()

putenv()

scandir()

setpgrp()

statfs(): Similar output is returned by *w_statfs.*, which requires the name of the MVS data set containing the file system to be passed as an argument. An application being ported will need to find a way to map the device number of the filesystem to the MVS data set name. One port used the system call *w_getmntent()* to retrieve file system information.

5.5.3 Functions that Behave Differently

access(): The syntax of the access() call is more restrictive in OpenEdition MVS than elsewhere. The function call, *access(path,0)* fails with an invalid parameter error on OpenEdition MVS, but other UNIXes will check for the existence of the file.

fopen(): The options for fopen() on OpenEdition MVS do not include the use of the "t" or text mode, as in the example below. Most other UNIX systems allow the "t" mode as a matter of upward compatibility, though it is a no-op.

```
fopen(path, "rt")
```

5.6 Use of MAKE and the Compiler

There are some differences between the make facility as implemented on OpenEdition MVS and as implemented on other UNIXes. The OpenEdition MVS make conforms to the requirements of the POSIX standards but tends to be less tolerant of nonstandard make file contents than other makes. The OpenEdition MVS make is being continually enhanced so that its behavior is more similar to these. Some of the items listed below may no longer be applicable on your system. The list comprises differences detected in our port and others and is provided as a guide. It should be emphasized that none of these differences caused any significant problems in any of the ports.

5.6.1 Explicit Casting

The OpenEdition MVS compiler is more restrictive regarding the use of different variable types than other compilers. Explicit casting is often necessary for this reason.

5.6.2 c89 Compiler Defaults to having the -o Option in the First Position

When using c89 to link, the compiler does not accept the *-o program* option at the end of the command.

```
c89 prog.o -o prog

Error message:

FSUM3008 Specify a file with the correct suffix (.c, .a, or .o),
or a data set name, instead of -o.
```

The link runs successfully when the -o option follows the compiler name.

```
c89 -o prog.o prog
```

In order to enable c89 to accept options after operands, do an export _C89_CCMODE=1 from the shell. This is documented in *OpenEdition MVS Command Reference*. The export command may be included in the $HOME/.profile.

5.6.3 No C Preprocessor

There is no C preprocessor (cpp on AIX, for example) and *c89 -E* behaves differently. Differences include:

- Comments are not removed.
- Checking is actually performed which can produce errors if the file does not contain C source.

5.6.4 Circular Dependencies

If the makefile contains a circular dependency in an unused part of the dependency tree, then OpenEdition MVS make issues an error. Other makes will tolerate this. It is particularly useful when writing code for multiple platforms where certain files will be built only for certain platforms.

5.6.5 Macro $ <

The meaning of special macro $< is different in OpenEdition MVS than in other UNIXes. The difference applies if a .h prerequisite file is outdated with respect to the object file. In this case, the $< macro is empty in the OpenEdition MVS make.

Circumventions would be either to replace $< with $*.c (or other appropriate source file extension) or to rebuild from scratch instead of “delta building.”

5.6.6 ar rc Not Supported

When running make invoked from a shell script, the `ar rc` command produces an error. Using `ar r` instead is successful.

5.6.7 Typing a Tab Character

If you are using the ISPF editor to edit your makefile, you cannot type a tab character (ISPF handles only displayable characters). Instead, you can:

1. Select a character that you will not be using in the file, for example, the character @.
2. At the beginning of each line of the recipe, type an @ instead of a tab character.
3. When you have finished editing the file, on the command line type:

   ```
   change @ X'05' all 1
   ```

   This converts the @ to the hex character 05, which is a tab. In ISPF Edit, the X'05' now displays as a blank space, which you cannot type over. If you use ISPF to edit or browse an existing file that has tabs in it:

   - In browse mode, the X'05' (tab) displays as a period (.) by default.
   - In edit mode, the X'05' displays as a blank space. When you edit the file, ISPF displays a message that the file contains unprintable characters and tells you how to use the FIND command to locate them. You can change the tabs back to an @ by typing the following on the command line:

     ```
     change X'05' @ all 1
     ```

5.6.8 Imakefile

Imakefile is not supported. An Imakefile must be converted to a makefile.

5.6.9 Miscellaneous

Root: A superuser with a UID=0 on OpenEdition MVS is not called *root* but shows up as the RACF user ID name. For example, the kernel generally is associated with a RACF user ID of *OMVSKERN*. It is recommended that the numeric UID 0 be used instead.

Threads: In the current release of OpenEdition MVS, while in a thread, you cannot use functions, such as *fork()*, *exec()*, and *system()*, that will cause another address space to be created.

File name truncation: If an MVS data set is searched for an include file, then the path specification is removed and the file name is truncated to eight characters. For example, *X11/IntrinsicP.h* and *X11/Intrinsic.h* both become INTRINSI. In order to resolve such conflicts, include statements must be changed.

System error texts: Other UNIX systems provide an external reference to a list of system error texts:

```
extern char *sys_errlist[];
```

On OpenEdition MVS this external reference remains unresolved. Use *strerror(errno)* instead of *sys_errlist*[*errno*].

Socket stream support: On OpenEdition MVS you cannot use runtime library stream functions (such as *fdopen(), fprintf(), fscanf()* and so on with sockets. This is significant when an application specifically uses it and also when you would like to redirect standard streams to sockets. A circumvention is to write “mini” runtime library functions to wrap around each function.

awk doesn′t handle whitespace: On OpenEdition MVS, *awk*, gives a syntax error if there is a whitespace between the function name and the left parenthesis in function calls. For example:

```
function do_macro (counter, file, ite_file)
```

has to be changed to:

```
function do_macro(counter, file, ite_file)
```

Linker not handling recursive references: OpenEdition MVS linker does not handle recursive references when object libraries have cross link dependencies. It is necessary to make multiple references to an archive library or other object library.

Unsigned character data type: The compiler default is to treat all character types as unsigned. You can override this by using a #pragma character directive in your code.

Chapter 6. Platform Specific Considerations

This chapter describes porting considerations that are specific to the OpenEdition MVS environment. Any port to OpenEdition MVS needs to address these areas. In each case, the background and the issues are outlined. Options and recommendations, where possible, are presented.

6.1 EBCDIC Character Encoding

Probably the most significant consideration when porting UNIX applications to OpenEdition MVS is that MVS uses a completely different set of hexadecimal character representations than UNIX systems. There is a requirement at times to translate between the two different systems and between variations of each. Often this is done automatically; sometimes it is a programmer′s responsibility.

Our experience was that all the required tools, tables and information are available. Programmers who are porting to OpenEdition MVS need to keep a clear head as to where they are and who is responsible for translation. When debugging, it is always worth asking yourself if the problem could be an ASCII/EBCDIC issue first.

6.1.1 Background

Given eight-bits to a byte, then there are 256 different possible bytes available for representing characters. Each different byte can then arbitrarily be assigned to a particular character. This process is called character encoding (or decoding).

IBM defined EBCDIC (Extended Binary-Coded Decimal Interchange Code) as one particular character encoding scheme for use in its computers and the American National Standards Institute (ANSI) defined a different code called ASCII (American National Standard Code for Information Interchange).

All UNIX and PC systems use ASCII in one form or another, but IBM mainframes, among others, continue to use EBCDIC. Therefore, when porting from any UNIX platform to OpenEdition MVS, ASCII and EBCDIC dependencies must be taken into account. This is a unique consideration for OpenEdition MVS and something to which even experienced porters will not be accustomed.

6.1.2 Code Pages

A code page for a specific character set determines the graphic character produced for each hexadecimal encoding. The code page used is determined by the programs and national languages being used. For internal processing, OpenEdition MVS uses EBCDIC. To be specific, it uses the character set in the *EBCDIC Latin 1/Open Systems Interconnection Code Page 01047*. Any text to be used in OpenEdition MVS shell processing must be converted to code page 01047. Depending on its origins, it could be in any one of a number of different code pages, both ASCII and EBCDIC. For example, a tar file would normally be stored using an ASCII code set. See *OpenEdition MVS User′s Guide* for more information.

6.1.3 Translation

Consider a number of different scenarios and who (that is, which software component) is responsible for translation in each.

Transferring files between a workstation and the file system: FTP handles the translation according to tables in TCP/IP. These can be customized if required and specified in the command or batch job invoking FTP.

Copying data between MVS datasets and the filesystem: The OCOPY, OGET, and OPUT TSO/E commands default to translating between the code page that the MVS system is using (known as the country-extended code page (CECP) and code page 01047. The *convert* option can be used to specify other tables.

Retrieving files archived on an ASCII system: The pax command with OpenEdition MVS, "explodes" an archive (pax, tar or cpio) into its individual files. Translation from ASCII to EBCDIC is specified by the *-o* option. For example the following shell command "explodes," translates and decompresses the archived file *courts.tar.z*

```
pax -o from=ISO8859-1,to=IBM-1047 -rzf courts.tar.Z
```

Accessing data in the HFS using NFS from another system: In text processing mode, data is converted between ASCII and EBCDIC by NFS. The default translation table (internal to NFS) converts between EBCDIC code page 0037 and ISO 8859-(ASCII). For accessing OpenEdition MVS files, the OEMVS311 translation table is specified either in the mount command or in the *xlat* processing attribute. User defined tables may also be specified. (see *DFSMS/MVS NFS Customization and Operation* for further information.) Note that in certain circumstances, when using the od command for example, translation would not be required. This can be achieved by specifying the *binary* option on the mount command. For example:

```
mount c1jes2: "/hfs/u/smitha,binary" /u/smitha.oe
```

Application to application communication: When two processes using different code pages communicate, it is the responsibility of the application to code the translation.

In the Sysdeco Innovation application, there is a requirement for the client process (running on an ASCII workstation) to communicate with the server process (running on an EBCDIC mainframe). Translation between the ASCII and EBCDIC code pages is the responsibility of the application. It is also the application's responsibility to determine which data needs to be translated. That is, text data is translated, binary data is not.

The Sysdeco Innovation application already has code to translate between different ASCII encodings on the client and the server. Thus it was a simple matter to code the EBCDIC translation. By using two tables (one for ASCII to EBCDIC and one for EBCDIC to ASCII), the code uses the character as the index into the table to look up the translated character. All that is required is a translation table between the two code pages (In our case, ASCII ISO 8859/1 and EBCDIC Code Page 1047).

The ASCII/EBCDIC translation table supplied in the sample libraries shipped with the TCP/IP for MVS software was used. The dataset name is:

```
TCPIP.V3R1.SEZATCPX(OEMVS311)
```

An alternative would have been to use the *iconv()* function. This provides translation between two specified character sets or code pages. Sysdeco Innovation chose to implement the translation on the client, and as *iconv()* is not available on all the Systemator supported client platforms, then it was not appropriate in our case.

6.1.4 Effect of ASCII/EBCDIC on Collating Sequence

Some functions will give different results in OpenEdition MVS than in ASCII based systems. When a function uses the relative position of characters to do a sort or compare, the results will differ depending on whether you are running on a platform with the ASCII encoding or running that same function on a platform based on the EBCDIC encoding. Examples are:

```
String compare   (strcmp (s1,s2))
String uncompare (strucmp (s1,s2))
Memory compare   (memcmp (s1,s2))
```

To illustrate in ASCII, when comparing the character “A” to the character “a,” “A”<“a,” but in EBCDIC, “a”<“A.” This will result in different return codes on compares and different output sequences on sorts.

It is useful in the early stages of a port to identify all usage of these commands. A code checker can be helpful. Decisions then have to be made as to whether to recode the logic of the application or to perform a translation immediately prior to affected usages.

In our case it was a simple matter to slightly alter the logic of the Systemator application in two places where strings are compared.

6.1.5 Working in the Shell When You Have Different Code Pages

Differences between the code page used internally by OpenEdition MVS and the code page used by a workstation or 3270 terminal can cause problems in entering and displaying certain characters.

Characters entered at a non-3270 workstation are normally translated by the 3270 emulation program into the CECP used by the MVS system. Any differences between the CECP and Code Page 01047 (the code page used by OpenEdition MVS; see 6.1, “EBCDIC Character Encoding” on page 39) need to be allowed for. In our case the CECP used was *U.S./Canada Country Extended Code Page 00037.* There are four characters with different representations. They are:

- Left square bracket ([)
- Right square bracket (])
- Circumflex (caret) (^)
- Not symbol (¬)

The square bracket characters are significant in both C code and shell scripts. It is therefore important that:

1. They are displayed correctly at the workstation being used.
2. The user has the ability to enter them from the workstation.
3. Printed output shows them correctly.

It is recommended that you determine the CECP used in your MVS installation (ask your friendly MVS systems programmer) and the differences between it and Code Page 1047.

6.1.5.1 Customizing a Terminal Emulator

When using workstations as 3270 emulators, we needed to change the code page used by the emulator so that it converted from the ASCII code of the workstation to the correct EBCDIC code for OpenEdition MVS. For example, Communications Manager/2 for OS/2 has the facility to use user-defined code pages. These are simply created by editing any IBM supplied code page. This process ensures that the correct characters will be displayed at the workstation. It may also be necessary to use the keyboard mapping functions of the workstation software to allow square brackets to be entered. Again, for example, Communications Manager/2 has such a function. It may also be necessary to check font tables if printing on your workstation.

For users of Communications Manager/2, a user defined code page is now available from IBM. This ensures the correct display of the square brackets and of the circumflex character, which is required when using the `bc` shell command. For details on obtaining this code page, refer to OpenEdition MVS information APAR #II08066.

6.1.5.2 Using the Convert Option on the OMVS Command

By default, the OMVS command uses a null character conversion table, (BPXFX100). This does no translation between code pages. The OMVS command has a CONVERT option that lets you specify a conversion table for converting between code pages. The table you want to specify depends on the code pages you are using in MVS and in the shell. For example, if you are using code page 0037 on your MVS system and code page 1047 in the shell, specify the following when you enter the OMVS command:

```
OMVS CONVERT((BPXFX111))
```

Conversion table BPXFX111 will display square brackets correctly for operations that are performed in the shell. For example, square brackets will be displayed correctly in a file that is processed with the `cat` shell command. Using the `ed` editor from the shell will also display the square brackets correctly if the correct CONVERT option is used. This technique works from both 3270 type terminals and workstations running a 3270 emulator. For more information, see the OMVS command description in *OpenEdition MVS: Command Reference*. At this time there are only conversion tables to handle conversion of code page 0037 to and from code page 1047. The next release of OpenEdition MVS, that will be packaged in MVS/ESA SP 5.2.2, will contain support to convert many more code page types.

6.1.5.3 Using NFS and the vi Editor

Using NFS, we were able to mount our HFS to our RS/6000 workstation. In our case, most editing was performed using the `vi` editor under AIX against this NFS mounted HFS. The display and entering of square brackets was handled correctly by NFS doing all the ASCII to EBCDIC translation. We found this to be the preferred method.

6.1.6 Future

As additional methods of attaching terminals to OpenEdition MVS are provided in the future, code page translation could become more and more complex. There is a recognized requirement to provide a cohesive set of user and program interfaces to manage the translation of ASCII and EBCDIC data on a session basis. The next release of OpenEdition MVS that is packaged with MVS/ESA SP 5.2.2 will provide these interfaces.

6.1.7 Summary

Translation to and from EBCDIC is a requirement unique to porting to some selected platforms including OpenEdition MVS. However, conceptually it is no different than translating between different ASCII character representations. So long as the application logic does not rely on a particular encoding for any character, then EBCDIC translation need not be a major issue.

Attention should be paid at installation time to ensuring that all required characters can be correctly entered, displayed and printed from all workstations.

6.2 Interactive Terminal Connection

There are fundamental differences between the end-user interface in traditional MVS applications and traditional UNIX systems. These differences result from fundamental differences in the terminal and related processor hardware characteristics of the respective platforms upon which these operating systems have historically run. The differences are between dumb ASCII screens and synchronous (block-mode) 3270 devices. These differences are pertinent to full-screen, interactive, text oriented applications. They are not relevant when using GUI oriented applications from workstations.

The differences manifest themselves in two main areas:

- The way (or mode) in which applications are coded
- Usability for keyboard users - end user, developer, administrator

6.2.1 Application Coding

This section considers the impact of the terminal interface on the types of applications that can be run and how they need to be coded.

First let us define some terms. For the purposes of this discussion the following are all equivalent terms:

- Raw mode
- Non-canonical mode
- Character mode

Their “opposites” are:

- Cooked mode
- Canonical mode
- Line mode

In this book we will use the terms line mode and character mode.

Line mode is how typical 3270 screens (and therefore typical MVS applications) operate. Input from a screen is passed to an application a line at a time when the Enter key or a program function key is pressed.

Character mode is how typical ASCII screens (and therefore typical UNIX applications) operate. Input from a screen is passed to an application a keystroke at a time.

In UNIX systems, character mode enables most interactive, full-screen, text-oriented applications (as opposed to GUI-oriented) to work. Typically, such applications require the following capabilities, among others:

- Cursor to skip automatically to next field on completion of field
- Tab from one field to another
- Hide display of password

In a UNIX system, these capabilities are provided within the application code itself. The application sees every character as it is typed and chooses, for example, whether to display ("echo") it on the screen (to provide password function) and where to position the cursor (to provide tabbing functions).

In an MVS system, a very different approach is adopted. Here, the application describes the screen layout. That is, the position of all the fields and the "attributes" associated with them. It is then the function of the screen itself to perform the functions listed above. The application sits idle waiting for the user to press *Enter* to send in the input fields.

So what is the significance of all this? OpenEdition MVS provides an interactive terminal connection through TSO/E. Most workstations or terminals from which a user can log on to TSO/E are supported by OpenEdition MVS. This includes VTAM connected 3270 terminals and programmable workstations with 3270 emulation, as well as workstations in a TCP/IP network that provide X3270, TN3270 client function or any other 3270 emulation. Only line-mode terminal operation is supported. It is for this reason that OpenEdition MVS does not currently support the `vi` editor. OpenEdition MVS cannot currently run applications written in character mode; neither can OpenEdition MVS applications access the 3270 datastream. Given an application written to the POSIX.1 general terminal interface (GTI), then character mode input is required. This is simply not an operating characteristic of 3270 terminals or workstations emulating 3270 terminals.

This effectively means that there are the following options:

1. Use a different application model - for example, client/server where the end-user part of the application is distributed to the workstation.
2. Find some way to allow a POSIX application to read and write the 3270 datastream so that it can exploit the function of 3270 devices.
3. Find some way to attach ASCII screens to OpenEdition MVS such that every keystroke can be responded to as it is pressed. That is, provide character mode terminal connection.

The first approach was the one used by Sysdeco Innovation and, given the current trends towards distribution of end-user presentation functions to the workstation, the approach increasingly likely to be adopted. However, IBM recognizes the needs of customers with existing networks of 3270 screens and

dumb ASCII screens and intends to provide solutions in all these areas in the next release of OpenEdition MVS that will be shipped with MVS/ESA V5.2.2.

6.2.2 Usability of the Terminal Interface

The terminal interface on 3270 and 3270-emulating screens is different than that found on traditional UNIX screens and also different than that found on GUI workstations. This can be a usability issue: users take time to get used to a different interface, especially when frequently moving between different environments.

For the Systemator port we mounted the OpenEdition MVS filesystem onto AIX and used an X-station as the programmer's workstation. This provided a familiar working environment and gave access to character mode applications, such as the `vi` editor.

Some of the usability issues that someone new to the 3270 interface will encounter are:

- Enter key is in a different position (Ctrl key).
- Other keys may be unfamiliar, may not be present or be implemented differently. It is recommended that the keyboard mapping functions of the screen and terminal emulator are used to best match what the user is comfortable with.
- Use of *escape* character to enter control characters; see *OpenEdition MVS User's Guide* for further details.
- Display of shell output is dependent on shell status; see 6.2.2.1, "Running/Input Mode" for further details.
- TSO/E commands can only be entered from the shell environment by pressing the TSO function key. When you are in TSO command mode, you cannot issue shell commands until you press the PA1 key to switch back into the shell.

  Note: A shell command `tso` is provided on an "as is" basis with this book to enable TSO commands to be entered normally from the shell and to be used within a shell script. This command will be incorporated into the next release of OpenEdition MVS.
- There is no `vi` editor available on OpenEdition MVS. This will be available in the next release of OpenEdition MVS that is shipped as part of MVS/ESA SP 5.2.2.
- OpenEdition MVS provides scrolling through use of PF keys. Later releases will provide cursor key scrolling.

6.2.2.1 Running/Input Mode

The status of a shell refers to the shell's ability to accept input or display output at a particular time and is shown at the bottom right of the screen. For full details, see *OpenEdition MVS User's Guide*.

During normal interactive shell processing of simple commands, the shell alternates between *INPUT* and *RUNNING*. The shell is only able to display output while in *RUNNING* status. However *RUNNING* status requires that OpenEdition MVS poll the shell. In order to avoid the overhead of continuously polling inactive sessions, a shell is moved from *RUNNING* to *INPUT* status after approximately 15 seconds of inactivity. This means that any output that has not

been displayed before the status change cannot be displayed. The user must press the Refresh function key (F10) in order to see the output.

This may be a serious usability issue for some users. We found it particularly frustrating while performing debugging and running lengthy `make`, `dbx`, and archive (`ar`) commands to have to keep pressing F10 to determine if the command had completed. Other ports have had similar experiences.

Note that the command is still being processed while the shell is in *INPUT* mode. It is just that no output will be displayed, so the user has no way of knowing when a long running command or shell script has completed except by regularly pressing F10.

A number of possible solutions to this issue have been discussed. They include:

- User option to alter the *RUNNING* mode time before *INPUT* mode
- Stay in *RUNNING* mode as long as a program is running, that is while the shell has a child process
- Increase TTY buffer pool size
- Simulate terminal activity to keep the shell permanently in *RUNNING* mode

Our preferred approach was to run a program in the background for each user. The program writes an alarm character to the terminal every few seconds (*NOALARM* is used when issuing the OMVS command to start the shell to silence the beeps). This prevents the shell from entering *INPUT* mode. It stays in *RUNNING* mode all the time.

Some sample code is given in Appendix D, “Staying in Running Mode” on page 79. This code is supplied on an “as is” basis. Implementation of this code may have a performance impact on OpenEdition MVS. If necessary, the code may be removed without affecting logged-in users.

6.3 Coding and Running Daemons Under OpenEdition MVS

Certain functions in most UNIX systems require that special privileges be set before a user is authorized to execute these functions. In OpenEdition MVS, because it uses the traditional MVS security products along with UNIX permission bit security, there are additional things that you need to be aware of. If your application performs a particular function that MVS deems to be privileged, your application will have to be executed out of a special MVS data set called an MVS authorized program library. Many MVS system functions, such as entire supervisor calls (SVC) or special paths through SVCs, are sensitive. Access to these functions must be restricted to authorized programs to avoid compromising the security and integrity of the MVS system. Programs that use the *setuid()*, *seteuid()*, and the *passwd()* functions have to be linkedited and executed from an MVS authorized program library.

Systemator has a daemon called *systord* that we had to port to OpenEdition MVS. Each Systemator client must enter their user ID and password, which is verified by the systord daemon running under OpenEdition MVS. The systord daemon uses the combination of the setuid, and passwd functions to authenticate the client. For more information on the *setuid()* and *passwd()* functions, see *OpenEdition MVS Supplement for rlogin and Associated Functions*. Because passwd is an authorized function we had to do the following:

1. Allocate an MVS data set called ITSC.AUTH.LOAD with the following attributes:

   ```
   DSORG = PO
   RECFM = U
   LRECL = 0
   BLKSIZE = 6144
   ```

   We allocated it large enough to hold our daemon program.

2. Change our makefile to link our daemon program into this MVS data set and into the HFS:

   ```
   # Minimum makefile for creating systord on MVS
   #
   CFLAGS=-I"//'OMVS.SFOMHDRS'" -Dmvs -D_OPEN_SOCKETS -DMVS
   OBJECTS=passwd.o systord.o users.o util.o
   systord:$(OBJECTS) systordmvs.o
   .$(CC) -o "//'ITSC.AUTH.LOAD(SYSTORD)'" $(OBJECTS)
   .$(CC) -o systord systordmvs.o
   ```

3. The OpenEdition file system has a unique feature that allows you to have a program executed out of an MVS data set rather than it being executed out of the HFS. This sticky bit feature, when set, directs OpenEdition MVS to search for the program in a MVS data set allocated to the user's STEPLIB, the system link pack area (LPA) or the system link list concatenation (LNKLST). If the program is not found in an MVS data set that is allocated to the user's STEPLIB, the system link pack area (LPA) or the system link list (LNKLST) concatenation, then it will be executed from the HFS. As an extra check, a small program coded with *printf()* statements was linked into the HFS with the same name as our daemon program that was linked into the MVS authorized library. If for some reason our daemon program got deleted out of the MVS authorized library or the sticky bit was not set correctly, the following would be displayed after trying to start the daemon:

   ```
   IBM
   Licensed Material - Property of IBM
   5655-068 (C) Copyright IBM Corp. 1993
   (C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
   (C) Copyright Software Development Group, University of Waterloo, 1989.

   All Rights Reserved.

   U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
   Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

   IBM is a registered trademark of the IBM Corp.

   /u/trygve/systemator: > systord
    Caution:
    This daemon program is being loaded from the hierarchical file system
    and should be loaded from an MVS authorized library.  Please check
    to make sure you have the daemon program linked into an MVS
    authorized library and you have the sticky bit set for this
    program in the file system.

   /u/trygve/systemator: >


    ===>

   ESC=# 1=Help    2=Subcmd 3=HlpRetrn  4=Top      5=Bottom   6=TSO
         7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve
   ```

   The sticky bit is set using the chmod shell command with the *t* permission. See *OpenEdition MVS Command Reference* for more information on the chmod

command. In our case, ITSC.AUTH.LOAD is part of the system link list (LNKLST) concatenation and part of the MVS authorized program facility (APF), so the daemon will be executed from this authorized library.

Note: You will need to discuss the copying of your program into an MVS authorized library with your MVS systems programmer.

4. RACF permit the OpenEdition MVS user ID that will be running the daemon to the BPX.DAEMON FACILITY class profile with the following command:

   ```
   PERMIT BPX.DAEMON CLASS(FACILITY) ID(SYSADM) ACCESS(READ)
   ```

   See *Planning: OpenEdition MVS* for more information on how to control security in an OpenEdition MVS environment.

6.3.1 How to Transport This MVS Load Module

During the port we successfully linked and started our daemon program from an MVS authorized library data set. If you are a UNIX vendor, you will probably want to copy this load module into the HFS with all your other executables to eventually be transported on some UNIX tape medium. The problem is not in copying the daemon program into the HFS; there are documented procedures in the *OpenEdition MVS User's Guide* to accomplish this. The problem is when you want to copy the daemon program back into the MVS authorized library data set. An executable file copied from the HFS into an MVS data set is not executable under MVS because some required directory information is lost during the copy to the partitioned data set (either PDS or PDSE). For our port, we decided to save the daemon's object files in the HFS and supply a makefile like the example in 6.3, "Coding and Running Daemons Under OpenEdition MVS" on page 46 to re-link the daemon program into the MVS data set at the customer's site.

Another way of transporting MVS load modules is to avoid the HFS altogether and use the TSO/E TRANSMIT and RECEIVE commands. The TSO/E TRANSMIT command has an option, OUTDA, that will transform the load module into a sequential data set; that enables you to use FTP with the BINARY option to copy the MVS load module to a workstation:

1. Execute the TSO/E TRANSMIT command:

   ```
   TRANSMIT C1JES2.WELLIE2 DA('ITSC.AUTH.LOAD(SYSTORD)') OUTDA('WELLIE2.UNLOAD')
   ```

   This creates a fixed block 80, sequential data set called WELLIE2.UNLOAD.

2. Run FTP from your workstation to copy this fixed block 80, sequential data set. Remember to use the BINARY option.

```
[C:\] ftp c1jes2
Connected to c1jes2.itsc.pok.ibm.com.
220-FTPSERVE IBM MVS V3R1 at C1JES2.ITSC.POK.IBM.COM, 09:37:50 on 02/14/95
220 Connection will close if idle for more than 5 minutes.
Name (c1jes2): wellie2
331 Send password please.
Password:
230 WELLIE2 is logged on.
ftp> binary
200 Representation type is IMAGE.
ftp> get 'wellie2.unload' systord.bin
200 Port request OK.
125 Sending data set WELLIE2.UNLOAD FIXrecfm 80
250 Transfer completed successfully.
local: systord.bin remote: 'wellie2.unload'
58800 bytes received in 0.59 seconds (96 Kbytes/s)
ftp>
```

3. At this point, the binary is stored on your workstation. You can copy it to a tape or diskette, but always remember to copy it as a binary file.
4. To send it to another OpenEdition MVS system from a diskette or a workstation, run FTP with the BINARY option.

   Note: You may need to use the TCP/IP FTP SITE command (see Appendix B, "Uploading the Source Files from the Diskette" on page 73 for an example) or preallocate a data set with the following attributes on your target MVS system before issuing the FTP PUT command:

   ```
   DSNAME=RCONWAY.UNLOAD.SYSTORD
   DSORG=PS
   RECFM=FB
   LRECL=80
   BLKSIZE=32720  (or some other multiple of 80)
   ```

```
[C:\] ftp wtscmxa
Connected to wtscmxa.itsc.pok.ibm.com.
220-FTPSERVE IBM MVS V3R1 at WTSCMXA.ITSC.POK.IBM.COM, 10:37:50 on 02/15/95
220 Connection will close if idle for more than 5 minutes.
Name (wtscmxa): rconway
331 Send password please.
Password:
230 RCONWAY is logged on.
ftp> binary
200 Representation type is IMAGE.
ftp> put systord.bin 'rconway.unload.systord'
200 Port request OK.
125 Storing data set RCONWAY.UNLOAD.SYSTORD
250 Transfer completed successfully.
local: systord.bin remote: 'rconway.unload.systord'
58800 bytes sent in 1.4 seconds (41 Kbytes/s)
ftp>
```

5. You now have to rebuild the load module with the TSO/E RECEIVE command:

```
  Menu  List  Mode  Functions  Utilities  Help
------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> RECEIVE INDA('RCONWAY.UNLOAD.SYSTORD')


  INMR901I Dataset ITSC.AUTH.LOAD from WELLIE2 on N1
  INMR902I Members: SYSTORD
  INMR906A Enter restore parameters or 'DELETE' or 'END' +
```

6.4 Other Function Not Currently Available with OpenEdition MVS

The following is a list of things that are available on some UNIX platforms but not currently available on OpenEdition MVS.

The list is not exhaustive nor is its order of any significance. It is provided as a checklist to help determine the feasibility or size the effort of moving applications to OpenEdition MVS.

If you have a particular requirement for any of these missing functions, please contact your IBM marketing representative.

- Korn shell only. Shell scripts that require other shells must be rewritten for the korn shell for now.
- NFS client support.
- C language only.
- Support for UNIX tape media.

Chapter 7. Migrating DB2 Programs

This chapter explains the process we used to port the source files that contained embedded SQL statements and were used when running the Systemator server function on a RS/6000 running DB2/6000.

The customer application used for testing the porting of the Systemator run time library uses DB2 as a tool for retrieving and storing data. As we said earlier, the Systemator run time library contains 850 object modules, but only one program, DBPGM, has the embedded SQL statements that are used to call DB2. Before the port, this code ran successfully on a RS/6000 with DB2/6000.

7.1 Structured Query Language

Structured Query Language (SQL) is a standardized language for defining and manipulating data in a relational database. You code SQL statements in your C source code to retrieve, update, delete, or insert data in a DB2 database. These SQL statements are treated like another language whose statements are embedded within a C language program. Before the program is submitted to a C language compiler, the SQL language statements are precompiled during the program preparation process. The precompiler replaces the SQL statements with calls that are recognized by the C language compiler. SQL statements can be static or dynamic.

7.1.1.1 Static and Dynamic SQL

Static SQL are SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statements do not change (although values of host variables specified by the statement might change). Static implies that the tables and fields to be used by the SQL statements in a program will not change from execution to execution of the program. This set of fields and tables used by a program is called a view. Static views can be precompiled and the DB2/MVS control blocks required for processing such a view can be reused for each and every execution of the program. DB2/MVS will even alter these views if the database is changed and the fields or position of the fields in the table change. In most cases, the compiled program will not have to be modified. Static SQL is used when writing highly optimized transaction programs where performance is an issue.

Dynamic SQL on the other hand are SQL statements that are prepared and executed within an application while the program is executing. In dynamic SQL, the SQL source is contained in host language variables rather then being coded into the application program. Dynamic SQL offers maximum flexibility at the expense of execution speed. Dynamic views do not have a consistent view of the tables and fields of a given DB2/MVS database, unlike static views. The fields to be used and the tables used will alter from execution to execution. Such programs will have SQL statements identified by the PREPARE or IMMEDIATE keyword on the EXEC SQL statement. Dynamic SQL is used for writing general database programming utilities or applications.

Even if a program has static views, DB2/MVS may consider it dynamic. DB2/MVS does this for non-CICS and non-IMS managed environments such as TSO and OpenEdition MVS. This is because DB2/MVS cannot as closely

cooperate with such private programs, to do so would jeopardize the security and integrity of DB2/MVS and possibly MVS/ESA.

7.1.2 SQL Syntax Differences Between Platforms

There are slight syntactical differences between SQL statements that run with DB2/6000 and those that run with DB2/MVS. The main differences that affected us during the port:

- With DB2/6000 it is possible to specify user ID and password on the CONNECT statement. The user ID specified was then used as a prefix on the unqualified DB2/6000 object names. With DB2/MVS it is not possible to specify user ID and password on the CONNECT statement. The security product that is running on the MVS system, for example RACF, is used to validate that the user ID has the authority to access the application server. Because of this, our application was changed from using unqualified names to using qualified names with DB2 objects. For more information, see 7.3.2, “Using Qualified and Unqualified Names with DB2 Objects” on page 56 and *DB2 Version 3: SQL Reference*.
- On the SQL CREATE TABLE statement, the name of the database where the table is created is not specified with DB2/6000 because the database name is specified on the SQL CONNECT statement. The opposite is true with DB2/MVS when using the SQL CREATE TABLE statement. The name of the database must be specified on the SQL CREATE TABLE statement because the database name cannot be specified on the SQL CONNECT statement. Examples of the SQL CREATE TABLE statement are shown in Figure 15 on page 53 and Figure 16 on page 54.
- A difference was also found when creating database indexes. With DB2/MVS, a unique index has to be created for the primary key in each DB2/MVS table. When creating this index, the SQL CREATE UNIQUE INDEX statement is used.
- When using dynamic SQL and coding the SQL PREPARE statement, a pointer to the SQL statement can be used with DB2/6000. This pointer points to a C string containing the SQL. On DB2/MVS this pointer could not be used. Instead a C structure containing two parts was used. The first element was a short integer and the second part a string. The short integer contained the length of the string that contained the SQL to be executed.

7.2 Migrating the Database Definitions

The Sysdeco programmer brought the DB2/6000 SQL statements which had been used for creating the DB2/6000 databases, tables and indexes. Some changes had to be made to these SQL statements because of syntax differences between DB2/6000 and DB2/MVS. We used the SPUFI (SQL processor using the file input) DB2/MVS facility to process these SQL statements on the MVS system. SPUFI is very easy to use and can be used for testing the SQL syntax using the SPUFI parameter AUTOCOMMIT(NO). Using SPUFI is also an easy way of testing that DB2/MVS definitions, like tables, are correct. For example, imagine that a DB2/MVS table named KUNDE has been defined, and the table is empty. The SQL statement SELECT * FROM KUNDE could be executed through SPUFI, and SPUFI will return SQL return code 100 indicating that end of file was reached in the KUNDE table. The return code 100 would also verify that the KUNDE table was built and processed correctly.

7.2.1 SQL Statements for the DB2/6000 Environment

This is an example of SQL used to define a subset of the DB2/6000 environment.

```
--***TBDBA***

CREATE DATABASE CANTAS
--     ON <database_location_path>
--     USING CODESET <codeset> TERRITORY <territory>
--     COLLATE USING <SYSTEM -or- IDENTITY>
    AUTHENTICATION SERVER
--     NUMSEGS <numsegs> SEGPAGES <segpages>
       ;

--***TBSQL***

connect to CANTAS in exclusive mode ;

create table AGENT(
       AGENT_NR smallint  not null ,
       A_NAVN varchar(20),
       PROSENT decimal(3,2),
       primary key (AGENT_NR)
       ) ;

create table AVSTAND(
       KUNDE_NR smallint  not null ,
       HAVN_NR smallint  not null ,
       ANTKM smallint ,
       AVSTBESK varchar(30),
       primary key (KUNDE_NR,HAVN_NR)
       ) ;

create table HAVN(
       HAVN_NR smallint  not null ,
       H_NAVN varchar(20),
       F_KOST decimal(7,2),
       LAND_ID char(3),
       primary key (HAVN_NR)
       ) ;

create table KUNDE(
       KUNDE_NR smallint  not null ,
       K_NAVN varchar(19),
       K_ADR varchar(40),
       K_TLF varchar(14),
       K_TLX char(10),
       K_TLFA char(8),
       LAND_ID char(3),
       primary key (KUNDE_NR)
       ) ;

--***TBINX***

connect to CANTAS in exclusive mode ;

create index KUNDEKM on AVSTAND (KUNDE_NR,ANTKM);
create index ZHAVAVSH on AVSTAND (HAVN_NR);
create unique index H_NAVN on HAVN (H_NAVN);
create index ZLANHAVL on HAVN (LAND_ID);
create index ZLANKUNL on KUNDE (LAND_ID);
```

Figure 15. SQL Statements for the DB2/6000 Environment

7.2.2 SQL Statements for the DB2/MVS Environment

This is an example of the SQL statements we used to define a subset of the DB2/MVS environment on MVS. The changes that we had to make compared to the SQL statements that were used for DB2/6000 are highlighted:

```
--***SQLDBA***

CREATE STOGROUP CANTASGR
       VOLUMES  ("DB2001")
       VCAT     DSN310
       ;

CREATE DATABASE CANTAS
       STOGROUP CANTASGR
       BUFFERPOOL BP0
       ;
--***SQLSQL***

create table AGENT(
       AGENT_NR smallint  not null ,
       A_NAVN varchar(20),
       PROSENT decimal(3,2),
       primary key (AGENT_NR)
       ) in DATABASE CANTAS;

create table AVSTAND(
       KUNDE_NR smallint  not null ,
       HAVN_NR smallint  not null ,
       ANTKM smallint ,
       AVSTBESK varchar(30),
       primary key (KUNDE_NR,HAVN_NR)
       ) in DATABASE CANTAS;

create table HAVN(
       HAVN_NR smallint  not null ,
       H_NAVN varchar(20),
       F_KOST decimal(7,2),
       LAND_ID char(3),
       primary key (HAVN_NR)
       ) in DATABASE CANTAS;

create table KUNDE(
       KUNDE_NR smallint  not null ,
       K_NAVN varchar(19),
       K_ADR varchar(40),
       K_TLF varchar(14),
       K_TLX char(10),
       K_TLFA char(8),
       LAND_ID char(3),
       primary key (KUNDE_NR)
       ) in DATABASE CANTAS;

--***SQLINX***

create index KUNDEKM on AVSTAND (KUNDE_NR,ANTKM);
create index ZHAVAVSH on AVSTAND (HAVN_NR);
create unique index H_NAVN on HAVN (H_NAVN);
create index ZLANHAVL on HAVN (LAND_ID);
create index ZLANKUNL on KUNDE (LAND_ID);

--***SQLINX1***

--***These are the extra SQL statements which was necessary with DB2/MVS .
create unique index HAVNX     on HAVN     (HAVN_NR);
create unique index AGENTX    on AGENT    (AGENT_NR);
create unique index KUNDEX    on KUNDE    (KUNDE_NR);
create unique index AVSTANDX  on AVSTAND  (KUNDE_NR,HAVN_NR);
```

Figure 16. SQL Statements for the DB2/MVS Environment

For a complete description of SQL on AIX, refer to publication *DB2 AIX/6000 SQL Reference*.

For a complete description of SQL on MVS, refer to publication *DB2 Version 3: SQL Reference*.

For a complete discussion of the differences regarding SQL implementation on different platforms, refer to the *Formal Register of Extensions and Differences in SQL*.

7.3 Coding the Connection to DB2/MVS from an Application

The most common way of executing DB2/MVS programs in non-CICS and non-IMS environments is to run the program under the DB2 TSO command processor called DSN. When executing a DB2 program in this way, the user executes the DSN TSO command which first sets up the DB2/MVS environment and then executes the program. This method of executing DB2 programs is not available under OpenEdition MVS.

7.3.1 The Call Attachment Facility

Another way of executing DB2 programs is by using the DB2 call attachment facility (CAF). When using the CAF interface, the DB2 program is executed directly, but the program must include some initialization and termination code for the DB2 execution environment. This is the method of executing DB2 programs from OpenEdition MVS. See *DB2 Application Programming and SQL Guide* for more information on the CAF interface.

The DB2 application programs used in this port had previously been running with DB2/6000. The call attachment facility does not exist in the DB2/6000 environment. Included below is a coding example showing the definitions, the entry code and exit code for using the CAF. The example should be changed to include values like the DB2 subsystem ID and the name of the plan from external sources instead of having them hardcoded in the program.

A complete listing of a program using the DB2 CAF interface is included in Appendix F, “Modified Version of DB2 IVP Program for Testing CAF” on page 85.

```
/***MVS & MVS OpenEdition CAF definitions*****************************/
/* Include external definitions                                      */
/*********************************************************************/
#pragma linkage(dsnali, OS)
#pragma linkage(dsntiar, OS)
/*********************************************************************/
/* Include C library definitions                                     */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/* Start main */
extern main()
{
/* ***CAF Start *** ********************************************** */
long int fnret;
char     opnfunc[13 ] = "OPEN ";
char     clsfunc[13 ] = "CLOSE ";
char     plan[9] = "DB2CPLAN "; /* name of DB2 PLAN******/
char     term_opt[5] = "SYNC";
long int return_code;
long int reason_code;
/* extern char     DB2SSID[]; */
char     DB2SSID[4] = "DB3B"; /****NAME of DB2 SUBSYSTEM**/
char     result_msg[64] = "***result_msg***";
int      *rpc_fault_status;
int      r_f_s;
/* ***CAF End*** ******************************************** */
printf("main start\n");
```

```
/* ***CAF Start Open the plans connections with DB2***************** */
rpc_fault_status = &r_f_s;
fnret = dsnali(opnfunc,
               DB2SSID,
               plan,
               &return_code,
               &reason_code);
printf("Open request fnret = %d\n", fnret);
printf("Open request return_code = %d\n", return_code);
printf("Open request reason_code = %d\n", reason_code);
fflush(NULL);
if (fnret != 0)
{
 sprintf(result_msg,
         "*** Unable to open DB2 for return: *",
         "%ld REASON: %ld",
         return_code,
         reason_code);
 *rpc_fault_status = 1;
 printf("main end\n");
 return;
}
/* ***CAF End*************************************************** */
/* ***CAF Start Close the plans connections with DB2***************** */
fnret = dsnali(clsfunc,
               term_opt,
               &return_code,
               &reason_code);
printf("Close request fnret = %d\n", fnret);
printf("Close request return_code = %d\n", return_code);
printf("Close request reason_code = %d\n", reason_code);
fflush(NULL);
if (fnret != 0)
{
 sprintf(result_msg,
         "*** Unable to close DB2 for return: *",
         "%ld REASON: %ld",
         return_code,
         reason_code);
 *rpc_fault_status = 1;
 return;
}
/* ***CAF End*** ************************************************** */
 printf("main end\n");
}
/* End main */
```

7.3.2 Using Qualified and Unqualified Names with DB2 Objects

In DB2 systems it is possible to use unqualified or qualified names on DB2 objects such as DB2 tables and views. For example, a DB2 table called TABLE1 would be an unqualified name, while DB2T.TABLE1 would be a qualified name. A DB2 table named DB2T.XXX.TABLE1 would also be a qualified name.

DB2 will use qualified names as they are, but unqualified names will always be prefixed by either a user ID or a SQL ID. The default is to use the user's user ID as the prefix, but through the use of SQL, the SQL ID can be used as a prefix. The SQL ID is the authorization ID that is used as the implicit qualifier of table, view, and index names in dynamic SQL statements. The SQL ID, along with the

other authorization IDs of a process, are used for authorization checking. The SQL ID could be specified as a parameter on the DSN BIND statement for static SQL and by using the SET CURRENT SQLID=USER for dynamic SQL.

The testing of the Systemator application was started against the generated CANTAS database. Problems arose during testing because the application was using unqualified names for DB2/MVS tables and views. The databases had been generated using user ID KJELLAA, which then was automatically used as a prefix for the DB2/MVS tables and views. During the testing of the application, another user ID, TRYGVE, was also used. User ID TRYGVE tried to access the unqualified tables, which had been established with user ID KJELLAA. This resulted in a SQL error code because now user ID TRYGVE was used as the prefix, but the only allowed prefix was KJELLAA.

When using DB2/MVS it is recommended to use qualified names. Up to this point, the Systemator application did not use qualified names. With DB2/6000 it is possible to specify a user ID and password on the SQL CONNECT statement. This user ID is then used as a prefix to the unqualified names. On DB2/MVS it is not possible to specify user ID and password on the SQL CONNECT statement. We found three possible solutions to our problem:

- Continue to use unqualified names and use a SQL statement in the application, which sets the prefix using the statement, SET CURRENT SQLID=USER. This was not a good solution for us because the execution of this statement requires SYSADM authority or could require some special customization of the security product that is running on the system.
- Continue to use unqualified names and use SQL to define synonyms. Using this method we would have to define one synonym for each user for each table. If we have 1000 users and 10 DB2/MVS tables we have to define 10000 synonyms. This is not a good solution from a practical point of view and it also creates a lot of information in the DB2/MVS catalog.
- Start using qualified names for DB2/MVS tables and views. This method will demand changes in our application but it is the solution that we chose. In our case this was not a large change because each user has to be authenticated by the server when they login, so our application has the user ID that can then be passed to DB2/MVS.

7.3.3 DB2/MVS Header Files

There are differences in the use of header files between DB2/6000 and DB2/MVS if the C language is used.

With DB2/6000, structures SQLDA and SQLCA are described in header files that reside in the file system. With DB2/MVS, these structures are imbedded in DB2/MVS load modules and cannot be examined until you run your source code through the DB2/MVS precompiler utility. Alternatively, you can see the description of the SQLDA and SQLCA structures in the *DB2 Version 3: Application Programming and SQL Guide*.

7.4 DB2/MVS Subsystem Overview

Figure 17 shows how you typically prepare a program that resides in a traditional MVS data set to run with DB2/MVS.

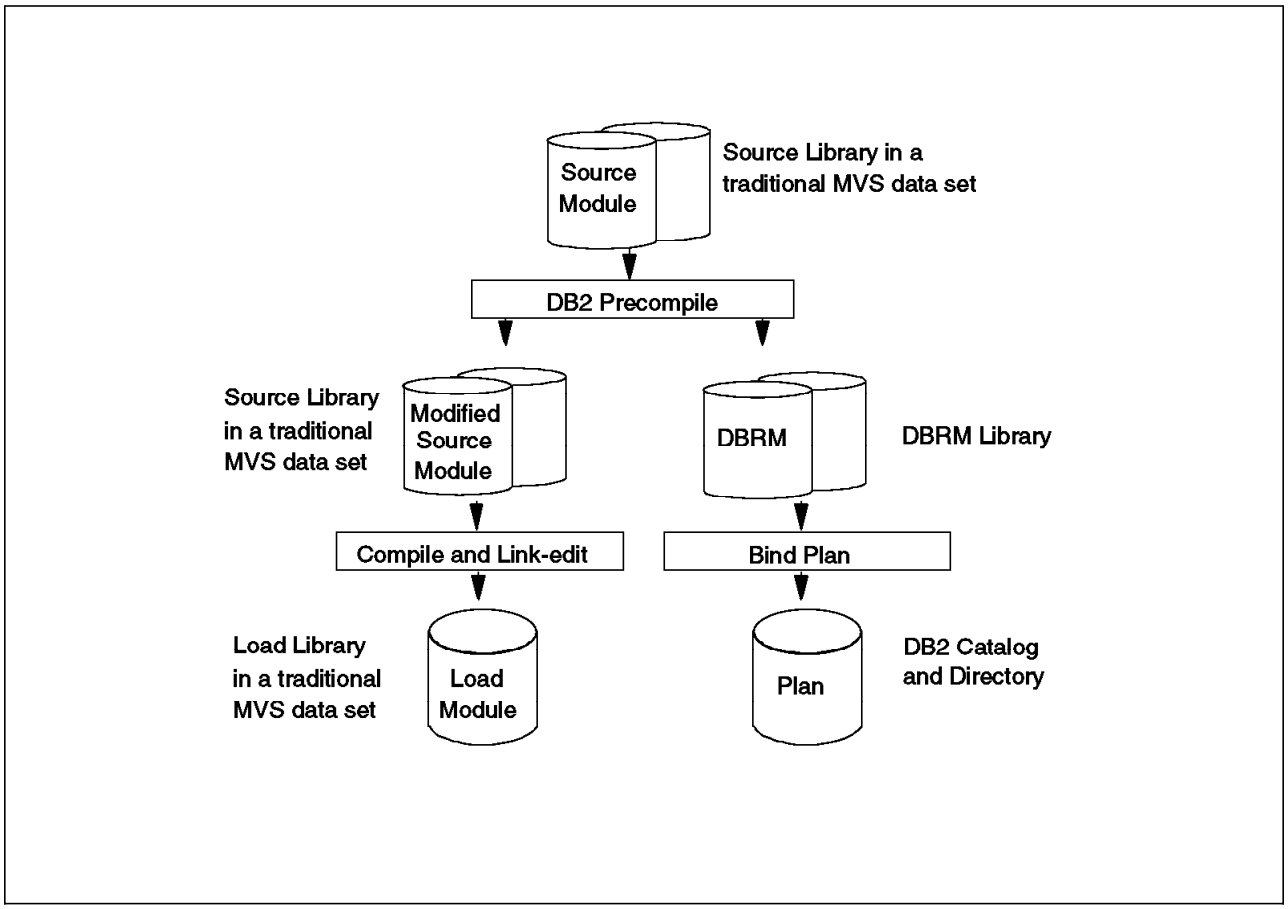


Figure 17. Flow of DB2/MVS SQL and Application Processing

When the source code is ready, the first step is to precompile the program with the DB2/MVS precompiler. The DB2/MVS precompiler takes the C source code with embedded SQL statements and generates two different data sets: the modified source module and the database request module (DBRM). In the modified source module, the EXEC SQL statements are replaced by a call to the DB2/MVS language interface. The DBRM contains all of the EXEC SQL statements that were extracted from the source program. Because one source program is separated into these two data sets, DB2/MVS inserts a consistency token into both data sets to check at run time that the two data sets originally came from the same source program. This consistency token is passed to the load module from the modified source module in the compiling and link-editing process.

The BIND process transforms the DBRM into DB2/MVS interpretable code. The output of BIND is called a plan. A consistency token is copied from the DBRM to the plan at that point. At execution time, this consistency token is checked against the consistency token of the load module.

In the bind/plan process, multiple DBRMs can be included in one plan. Because one DBRM consists of one program, a plan can contain information on SQL

statements for multiple programs. When a program is a common routine shared by multiple applications, one DBRM is bound into multiple plans.

7.4.1 Preparing a Program For DB2/MVS from the OpenEdition MVS Shell

As we said before, a program that has embedded SQL statements must be run through the DB2/MVS precompiler. There are two things to be considered:

1. Since the hierarchical file system is fairly new on MVS, DB2/MVS, in its current release, doesn't have the ability to read source code directly from the HFS as input to the DB2/MVS precompiler. It must be copied into a traditional MVS data set first. A REXX EXEC was written and is supplied with this book that will do all the copying of the C source from the HFS to a MVS data set along with executing the DB2/MVS precompiler.
2. All the C source files that contain embedded SQL statements, must fit within 80 columns in the MVS data set to be accepted by the DB2/MVS precompiler (the default statement length is 72 characters, but it is possible to increase this length to 80 characters using the MARGINS parameter of the DB2/MVS precompiler). This 80 character limit doesn't exist with files stored in hierarchical file systems in OpenEdition MVS and in UNIX. This restriction caused us to edit all the C source code that contained embedded SQL statements to fit within 80 columns. In our case this required us to edit a program that contained 5000 lines of code. In other cases this may affect many source files and many more lines of code.

7.4.2 Running the DB2/MVS Precompiler from the OpenEdition MVS Shell

The only documented ways to run the DB2/MVS precompiler is from clists or using the SFUFI facility from TSO/E or running a batch job. For those who prefer, we have written a REXX EXEC that can be run from the shell. The functions of the EXEC are as follows:

1. Copies the "C" source from the HFS files into a MVS data set
2. Runs the DB2/MVS precompiler and binds to a plan
3. Copies the output back into the HFS

Figure 18 on page 60 shows, within the dotted lines, the operations that are performed by the REXX EXEC. The EXEC, together with usage instructions, are supplied in Appendix G, "REXX EXEC to Run the DB2 Precompiler and BIND from the OpenEdition MVS Shell" on page 95.

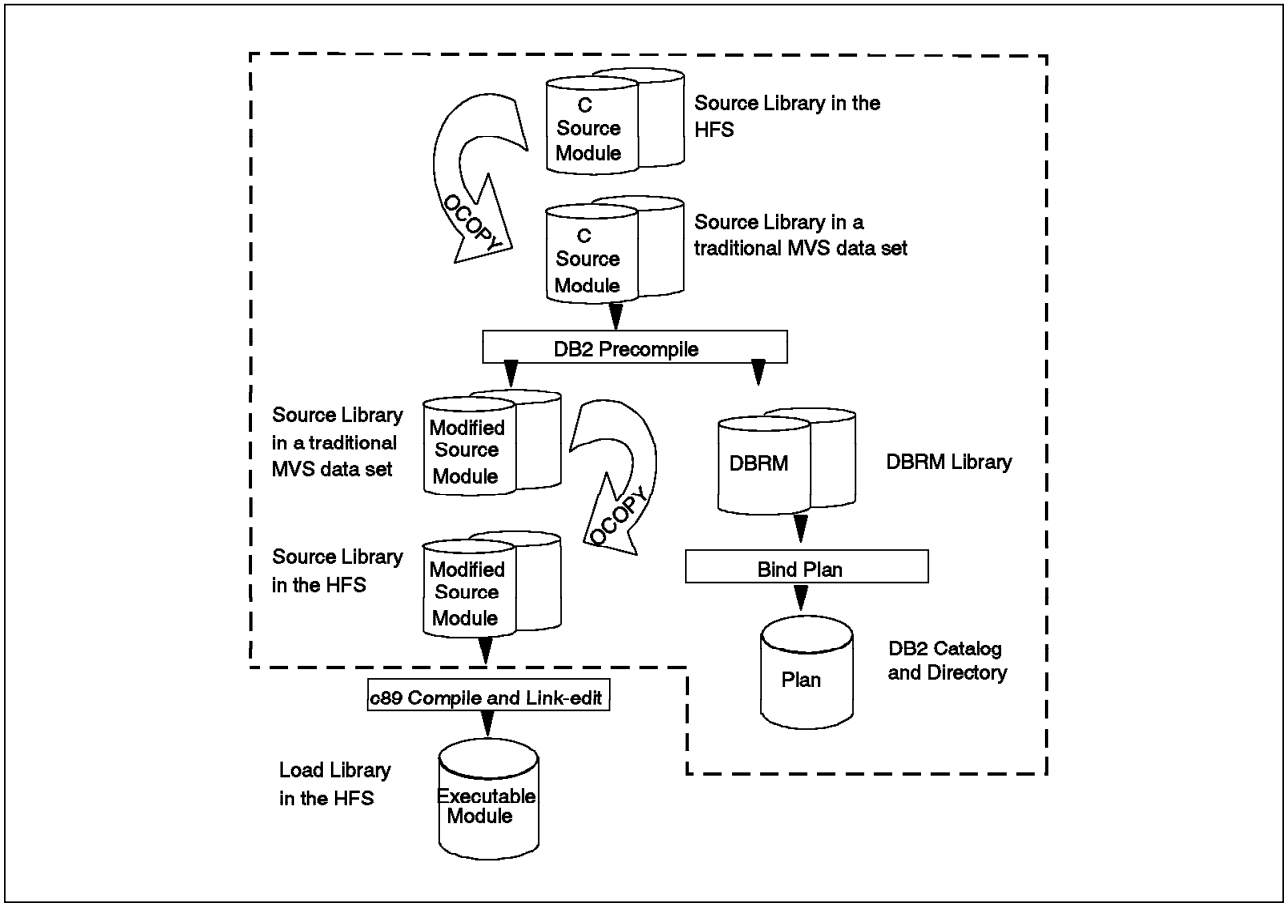


Figure 18. Flow of DB2/MVS SQL and Application Processing with OpenEdition MVS

7.4.3 Example of Executing a DB2/MVS Program Under OpenEdition MVS

Appendix F, “Modified Version of DB2 IVP Program for Testing CAF” on page 85, contains a copy of the sample C code that is shipped as part of the installation verification procedure (IVP) for DB2/MVS Version 3. The CAF example from 7.3.1, “The Call Attachment Facility” on page 55, has been added to this code so that it can be run from the OpenEdition MVS environment.

The DB2/MVS subsystem was fully functional on our OpenEdition MVS system before we started the port. All the customization including the steps necessary to get the installation verification procedures (IVPs) working was already done. Since we had limited DB2/MVS experience, it was decided that we should work on getting the IVPs working, using the DB2/MVS CAF interface under OpenEdition MVS, to gain some experience with DB2/MVS on a smaller scale, before trying to port the Systemator code. Refer to *DB2 Administration Guide, Volume I* for more information on the Installation Verification Procedures.

As a part of the DB2/MVS IVP setup, there are a series of jobs that need to be run to load the DB2/MVS database environment with data to allow the IVP to function. The IVP job DSNTEJ1 creates all the objects (storage group, databases, table spaces, tables, indexes, and views) and loads the sample tables used by our sample program. See the *DB2 Administration Guide, Volume I* for more information on preparing the DB2/MVS environment.

The IVP job, DSNTEJ2D, is a phone application written in the C language against the tables created by job DSNTEJ1. The DSNTEJ2D IVP job uses two programs, called DSN8MDG and DSN8BD3, which are link edited into one executable program. The sample code in Appendix F, "Modified Version of DB2 IVP Program for Testing CAF" on page 85, are the same two programs, DSN8MDG and DSN8BD3, that have been combined and the necessary CAF interface code added.

The DB2/MVS precompiler and the DB2 DSN BIND process is not available as a command from the OpenEdition MVS shell. A REXX EXEC called PRECOMP was written as a tool for running the DB2/MVS precompiler and BIND from OpenEdition MVS. Refer to Appendix G, "REXX EXEC to Run the DB2 Precompiler and BIND from the OpenEdition MVS Shell" on page 95, for more information on the PRECOMP REXX EXEC. This is how the program was run in the OpenEdition MVS environment:

1. Copy the sample program from the diskette that is supplied with this book (See Appendix B, "Uploading the Source Files from the Diskette" on page 73), and load it into a directory in the HFS. It should be called db2cpgm2.i.
2. Check the contents of line 18 in db2cpgm2.i. This line should be:
   - #define OMVS 1 /* ***if running the program under OpenEdition MVS*** */
   - #define OMVS 0 /* ***if running the program under MVS*** */
3. Check the contents of line 182 and 187 in db2cpgm2.i. These two lines contain the name of the DB2/MVS plan the program intends to use and the name of the DB2/MVS subsystem under which the program should run.
   - Change DB2CPLAN to the name of the PLAN.
   - Change DB3B to the name of the DB2/MVS subsystem.
4. The PRECOMP REXX EXEC was run from the OpenEdition MVS shell using the `tso` shell command. See Appendix G, "REXX EXEC to Run the DB2 Precompiler and BIND from the OpenEdition MVS Shell" on page 95, for more information on the PRECOMP REXX EXEC. See Appendix C, "OpenEdition MVS Shell Command to Run TSO Commands" on page 77, for more information on the `tso` shell command. This EXEC runs the sample C source code through the DB2/MVS precompiler, and copies the precompiled source code, which is output from the DB2/MVS precompiler into a file in your HFS directory.
5. Figure 19 on page 62 shows the running of the DB2/MVS precompiler from the shell with db2cpgm2.i as input. The translated source is stored in db2cpgm2.c. The DBRM is bound to the plan named DB2CPLAN.

   Note: To run this from the shell, the DB2/MVS library, SDSNLOAD must be used as a STEPLIB if it is not part of the MVS LNKLST concatenation. Add the following to your $HOME/.profile:

   ```
   STEPLIB=your_hlq.SDSNLOAD

   export STEPLIB
   ```

   Replace your_hlq with the high level qualifier that is used to prefix the DB2/MVS library, SDSNLOAD on your MVS system.

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/kjellaa: >



 ===> tso precomp pa=/u/kjellaa/ in=db2cpgm2.i ou=/db2cpgm2.c
       pl=db2cplan
```

Figure 19. Running the DB2/MVS Precompiler from the Shell

Alternately, the PRECOMP REXX EXEC can also be run from the shell by pressing the program function key that is setup to switch to TSO/E.

Figure 20 on page 63 shows the output from the PRECOMP REXX EXEC. The translated C source code is copied back into the HFS after the DB2/MVS precompiler runs.

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/kjellaa: >


precomp pa=/u/kjellaa/ in=db2cpgm2.i ou=db2cpgm2.c pl=db2cplan
PRECOMP started
IKJ56247I FILE SYSPRINT NOT FREED, IS NOT ALLOCATED
IKJ56247I FILE SYSTERM NOT FREED, IS NOT ALLOCATED
p.1=pa=/u/kjellaa/
p.1=in=db2cpgm2.i
p.2=ou=db2cpgm2.c
p.3=pl=db2cplan
pathi='/u/kjellaa/db2cpgm2.i'
patho='/u/kjellaa/db2cpgm2.c'
IDC0550I ENTRY (A) KJELLAA.DB2.PREINP DELETED
IDC0550I ENTRY (A) KJELLAA.DB2.PREOUT DELETED
OCOPY C input source from HFS
OCOPY DB2 PRECOMPILE
OCOPY ended with RC  0
DB2 precompiler ended with RC = 0
DSNT200I #  BIND FOR PLAN DB2CPLAN SUCCESSFUL
DB2 DSN BIND ended with RC = 0
OCOPY C translated source to HFS
ocopy ended with RC = 0
PRECOMP ended with highest RC = 0
/u/kjellaa: >
 ===>
```

Figure 20. Output from the PRECOMP REXX EXEC

6. The user then runs c89 to compile, prelink and link the db2cpgm2.c program.

   The CAF interface module DSNALI must be included in your executable during the link-edit step. All DB2/MVS interface modules reside in the DB2/MVS data set SDSNLOAD. In this library there are two modules, DSNELI (DB2 TSO attachment), it's alias (DSNHLI) and DSNALI (DB2 call attachment facility), and it's alias (DSNHLI1). Both the DB2 TSO attachment interface and the call attachment interface modules contain the same entry point declaration for DSNHLI. We need a way of telling the linkage editor to select the call attachment interface module instead of the DB2 TSO attachment interface. Ordinarily, when you run the linkage editor in a MVS batch job, you can pass an INCLUDE control card to tell the linkage editor to explicitly include the DB2 call attachment module DSNALI. With c89, there is no way to pass these control cards to the linkage editor. You must make a copy of the SDSNLOAD data set and manually delete the DB2 TSO attachment interface modules.

   The library DSN310.SDSNLOAD.NEW in the example below is a copy of our original DB2/MVS library, DSN310.SDSNLOAD. We deleted module DSNELI

and its alias DSNHLI from DSN310.SDSNLOAD.NEW so that our external references were resolved from DSNALI and DSNHLI1 respectively.

- Figure 21 shows how to run c89 to compile and link the sample program.

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/kjellaa: >



 ===> c89 db2cpgm2.c -l "//'DSN310.SDSNLOAD.NEW'"
```

Figure 21. Running c89 to Compile and Link the DB2/MVS Program

7. Create an HFS file named cardin in your current directory, which contains input parameters for the program. This is the same file that is shipped with the DB2/MVS IVP job DSNTEJ2D, which is supplied with the DB2/MVS product. The cardin file must be a file in the directory where the executable resides:

   ```
   L*
   LJO*
   L%SON
   LSMITH
   LBROWN         ALAN
   LBROWN         DAVID
   U                          0002304265
   ```

8. Figure 22 on page 65 shows the sample program being executed from the shell. Execute the program that is named a.out since the *-o* option was not used on the earlier c89 command. The STEPLIB environmental variable must be set before you run a.out to add the DB2/MVS library, SDSNLOAD, if it is not part of the MVS LINKLST concatenation. See *OpenEdition MVS User's Guide* for more information on using STEPLIB.

```
IBM
Licensed Material - Property of IBM
5655-068 (C) Copyright IBM Corp. 1993
(C) Copyright Mortice Kern Systems, Inc., 1985, 1993.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

/u/kjellaa: > a.out
main start
Open request fnret = 0
Open request return_code = 0
Open request reason_code = 0
Close request fnret = 0
Close request return_code = 0
Close request reason_code = 0
main end
/u/kjellaa: >




 ===> a.out
```

Figure 22. Executing the Sample DB2 Program

9. The output from the program is in a HFS file called *report* that will be in your current directory. DB2/MVS SQL messages will also be found in this file.

Chapter 8. Other Considerations for a Complete Port

The focus of the work described in this book was to port the function of the Systemator application to the OpenEdition MVS platform. Other sections of this book concentrate on how that was achieved.

However, when porting applications to a production environment, there are other considerations to keep in mind. It is not sufficient merely that the code executes on the new platform. The ported application must be robust, secure, meet Service Level Agreements and so on. It is not the function of this book to tell you how to run your project or how to manage your installation, but the following sections are provided by way of a simple checklist for use when conducting a porting project and when bringing a ported application into production.

8.1 Project Management of the Port

When initiating a porting project, the following plan may help:

1. Set up a formal project and agree on milestones, completion criteria, subprojects and tasks, change control mechanisms, issue handling mechanisms, problem reporting mechanisms and so on.
2. Design and establish the hardware and software environment for the port. Consider OpenEdition MVS server, compilers, libraries, utilities, network access from workstation and tape facilities.
3. Allow for “teething” problems in establishing the environment.
4. Identify personnel to carry out the port. You will need skills in the structure, the maintenance, and the use and implementation of the application.
5. Provide access to people with MVS skills.
6. Size the port - produce a list of all function calls and identify any not supported on OpenEdition MVS.
7. Consider use of a “code checker.”
8. Transfer source files, procedures and data files to OpenEdition MVS.
9. Rewrite source-compile procedures, for example, change makefiles to use OpenEdition MVS facilities.
10. Iteratively compile each module, persistently and thoroughly eliminating all errors.
11. Create executable programs.
12. Rewrite program runtime procedures to use MVS facilities.
13. Test the application on OpenEdition MVS.
14. Integrate the ported application into the production MVS environment.

8.2 Systems Management Considerations

The final item in the list above refers to the additional considerations beyond code porting in order to provide a production service. The first section below describes general systems management disciplines that need to be in place. The second section outlines some specific considerations available on OpenEdition MVS.

8.2.1 General

In moving a UNIX application to MVS, you are not only changing the operating system, but you are changing the whole environment or culture in which the application will be run and managed. You are most likely bringing a new application into a world that already has a very different set of ideas about how to provide an IT service. The following questions will need to be considered:

- What are the performance requirements?
- Who is responsible for monitoring performance?
- What are the availability requirements?
- What are the security requirements?
- Who is responsible for managing change?
- Who is responsible for capacity planning?
- What are the backup requirements? Who is responsible for running backups?
- Who is responsible for recovery?
- What are the disaster recovery requirements?
- How will the introduction of code changes be managed?
- How will the distribution of code to workstations be managed?
- Who is responsible for storage management?

The answers to these questions may be very obvious. It may well be “business as usual.” However it may be much more complex, and have organizational and political aspects. The mainframe world may have much stricter controls over what can be done, when and by whom. These security controls need to be understood and planned for. For example, it may not be possible to login as root or to reboot the system to correct minor problems.

8.2.2 OpenEdition MVS Systems Management

OpenEdition MVS has access to all the systems management functions of MVS. In MVS terms, the HFS is another data set. Therefore, that data set can be managed just like any other MVS data set. The same facilities for security, recovery, backup, storage management, automation, job scheduling, performance management, network management, problem management and so forth are available. This is precisely the strength of OpenEdition MVS; the combination of an open application programming interface together with the industrial strength systems management capabilities of MVS/ESA.

Once storage management requirements for the new application have been agreed upon, they can be implemented through Systems Managed Storage (SMS). SMS will operate at the data set level, that is each filesystem is treated as one entity. For backup and archive purposes this may not be sufficient. A future release of ADSM (ADSTAR Distributed Storage Manager) will include a

new ADSM client component for OpenEdition MVS. This new client component will support the backup and archiving of individual files from the OpenEdition HFS (Hierarchical File System) to the storage resources that are under the control of the ADSM server component. This gives the capability to perform incremental backups where only changed files are backed up and to restore individual files in cases where one file has been accidentally deleted for example.

8.2.2.1 Performance Considerations

The first release of OpenEdition MVS concentrated on providing the POSIX APIs and provided a product that allowed customers to investigate the function provided, to port some applications and to assess the applicability of OpenEdition MVS to their environment.

Subsequent releases, while continuing to provide further standards compliance, place more emphasis on improving performance. For example, multiple processes within an address space was added to OpenEdition MVS with MVS/ESA V5.1. It is IBM′s intention that the next release of OpenEdition MVS will contain many more performance enhancements.

It is recommended that, if possible, installations plan to use the latest available release and maintenance level.

Our experience was that, during porting, the most important aspect of performance is response time. This directly determines the productivity of the programmer. We found that the most significant impact on our response time was the network configuration. We recommend that TCP/IP be used over SNALINK only for investigative purposes. We also recommend that OpenEdition MVS APAR OW07648 together with its corequisite TCP/IP APAR PN65950 be installed. For a production environment we recommend use of a 3172 or an Open Systems Adaptor in order to connect LAN-attached workstations to the mainframe. See Chapter 3, “System Configuration” on page 7, for more information.

8.2.2.2 OpenEdition MVS Performance Guidelines

The following is a list of general guidelines for improving the performance of OpenEdition MVS with MVS/ESA V5.1. Your MVS system programmer should implement these recommendations.

1. Give each user a separate HFS data set.
2. Use the “sticky bit” feature to have the shell (*sh*) and other commonly used commands execute out of the link pack area (LPA). See *Planning: OpenEdition MVS* for more information.
3. Put the following run-time library routines in the LPA:

Loaded for each C program	Loaded for each BPX utility
CEE.V1R3M0.SCEERUN(CEEEV003 with alias EDCZV)	SYS1.LINKLIB(BPXEV003 with alias BPXZV)
(CEEOLVD)	(BPXOLVD)
(CEEPLPKA)	(BPXPLPKA)
(EDCOV)	(BPXOV)
(EDCZ24)	(BPXZ24)
(CEEMMS)	(BPXMMS)

```
(CEEBINIT)                          (BPXBINIT)
(EDCUCSNM)
(EDC$LCNM)
(CEEEV012 with
alias FOMQVTV)
```

4. To improve compile performance by avoiding frequent loads for c89, put the following modules in LPA:

```
BPXOVI          EDCDP
EDCDC120        EDCDT
EDCDO           EDCPRLK
```

5. Put the rest of the LE/370 run time library in the virtual lookaside facility (VLF).

6. Add the following to the COFVLFxx member of parmlib to enable caching of UID and GID information:

```
CLASS NAME(IRRUMAP)
  EMAJ(UMAP)
CLASS NAME(IRRGMAP)
  EMAJ(GMAP)
CLASS NAME(IRRGTS)
  EMAJ(GTS)
CLASS NAME(IRRACEE)
  EMAJ(ACEE)
```

7. Avoid STEPLIB propagation by adding the code below to /etc/profile or $HOME/.profile

```
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec sh -L
fi
```

8. Ensure no DASD contention for header files (EDC.V1R2M0.SEDCDHDR) or run-time library stubs (CEE.V1R3M0.SCEELKED).

Appendix A. Code Checker Utilities

"Code checker" is a generic term for tools that analyze source code. There are a range of products available. The simplest perform *lint*-like error detection. The more sophisticated are themselves programmable and can be used to:

- Identify code that is not portable to or from a particular environment
- Monitor compliance with programming standards
- Quantify user-defined measures of complexity and maintainability

It is the first function that we are interested in here. A code checker could be used in advance of a porting project to identify code that will need to be changed. Ideally the tool would be able to detect noncompliance with POSIX standards, with XPG4, with UU (Universal UNIX) and with particular releases of OpenEdition MVS. In some cases, code checkers may also be able to provide replacement code.

However, in many cases, the reason that a particular piece of code does not port to a new environment has more to do with the way that a function has been used rather than the function itself. A code checking utility cannot easily determine the logic of an application. A utility that identifies all occurrences of a particular function will provide a useful checklist but will not be able to provide a complete or a well targeted list. A programmer familiar with the application and with the porting process is likely to produce a more useful list.

In our port, the programmer did not wish to use a code-checker. His familiarity with the source code enabled him to identify and target easily those sections of code that would not port.

The conclusion from our experience is that code checkers can provide a useful checklist if no one with porting experience or application knowledge is available. As more experience of porting to OpenEdition MVS is gained and collated, it may well become possible to program a code checker to automate at least part of the porting process.

Appendix B. Uploading the Source Files from the Diskette

The diskette distributed with this redbook contains the source code that is described in this book. The diskette contains the following files:

```
README   TXT     3830           <disclaimer information>

 Directory of BINARY            <data in binary format>
TSOBIN   CMD    58800           <tso command load module>
TSOBIN   DOC    98080           <tso command documentation>

Directory of TEXT               <data in text format>
POLL     C        461           <poll source>
DB2CPGM2 I      20401           <DB2 IVP source>
CARDIN            575           <DB2 IVP card input>
PRECOMP  REX    10656           <DB2 precompiler REXX EXEC>
HDRCPY   REX     9430           <REXX EXEC to copy header files>
```

B.1 Uploading the Source Files to the MVS Host

You can copy the sample code from the supplied diskette to the HFS a number of different ways, including:

- Using the Network File System feature if you have NFS installed on both your client workstation and the MVS host
- Using the File Transfer Protocol (FTP) facility of TCP/IP when both the workstation and the MVS host have TCP/IP installed
- Using the SEND and RECEIVE commands shipped with many 3270 emulator packages, including Communication Manager for OS/2

The most important thing to remember when uploading the source code is to make sure the binary data in the BINARY directory is not translated. As an example, the steps required to upload the source code using TCP/IP FTP from an OS/2 workstation:

```
[C:\]ftp c1jes2
IBM TCP/IP for OS/2 - FTP Client ver 09:44:28 on Mar 04 1994
Connected to c1jes2.itsc.pok.ibm.com.
220-FTPSERVE IBM MVS V3R1 at C1JES2.ITSC.POK.IBM.COM, 14:00:37 on 03/24/95
220 Connection will close if idle for more than 5 minutes.
Name (c1jes2): wellie3
331 Send password please.
Password: .......
230 WELLIE3 is logged on.
ftp> site blk=32720 lrecl=80 recfm=fb tracks unit=sysallda primary=10
200 Site command was accepted
ftp> binary
200 Representation type is IMAGE.
ftp> put a:\binary\tsobin.cmd 'wellie3.tsobin.cmd'
200 Port request OK.
125 Storing data set WELLIE3.TSOBIN.CMD
250 Transfer completed successfully.
local: a:\binary\tsobin.cmd remote: 'wellie3.tsobin.cmd'
58800 bytes sent in 3.6 seconds (15 Kbytes/s)
ftp> put a:\binary\tsobin.doc 'wellie3.tsobin.doc'
200 Port request OK.
125 Storing data set WELLIE3.TSOBIN.DOC
250 Transfer completed successfully.
local: a:\binary\tsobin.doc remote: 'wellie3.tsobin.doc'
98080 bytes sent in 5.9 seconds (16 Kbytes/s)
ftp> ascii
200 Representation type is ASCII.
ftp> put a:\text\poll.c 'wellie3.poll.c'
200 Port request OK.
125 Storing data set WELLIE3.POLL.C
250 Transfer completed successfully.
local: a:\text\poll.c remote: 'wellie3.poll.c'
461 bytes sent in 0.19 seconds (2 Kbytes/s)
ftp> put a:\text\db2cpgm2.i 'wellie3.db2cpgm2.i'
200 Port request OK.
125 Storing data set WELLIE3.DB2CPGM2.I
250 Transfer completed successfully.
local: a:\text\db2cpgm2.i remote: 'wellie3.db2cpgm2.i'
20400 bytes sent in 1.5 seconds (13 Kbytes/s)
ftp> put a:\text\cardin 'wellie3.cardin'
200 Port request OK.
125 Storing data set WELLIE3.CARDIN
250 Transfer completed successfully.
local: a:\text\cardin remote: 'wellie3.cardin'
575 bytes sent in 0.19 seconds (2 Kbytes/s)
ftp> put a:\text\precomp.rex 'wellie3.precomp.rexx'
200 Port request OK.
125 Storing data set WELLIE3.PRECOMP.REXX
250 Transfer completed successfully.
local: a:\text\precomp.rex remote: 'wellie3.precomp.rexx'
10656 bytes sent in 0.63 seconds (16 Kbytes/s)
ftp> put a:\text\hdrcpy.rex 'wellie3.hdrcpy.rexx'
200 Port request OK.
125 Storing data set WELLIE3.HDRCPY.REXX
250 Transfer completed successfully.
local: a:\text\hdrcpy.rex remote: 'wellie3.hdrcpy.rexx'
9430 bytes sent in 1.1 seconds (8 Kbytes/s)
ftp>
```

B.2 What to Do with the Source Files?

There are a few extra things that need to be done once all the source files are uploaded to the host. The tso command that is shipped in the tsobin.cmd and tsobin.doc files is in a special format. You must use the TSO/E RECEIVE command to rebuild the tso command and all its supporting pieces. Enter the TSO/E RECEIVE command with the INDA parameter for both the TSOBIN.CMD data set and the TSOBIN.DOC data set. Examples follows:

```
  Menu  List  Mode  Functions  Utilities  Help
------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> RECEIVE INDA('WELLIE3.TSOBIN.CMD')


 INMR901I Dataset POSIX.SCHOEN.TSOCMD.LOAD from SCHOEN on PLPSC
 INMR906A Enter restore parameters or 'DELETE' or 'END' +
```

```
  Menu  List  Mode  Functions  Utilities  Help
------------------------------------------------------------
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> RECEIVE INDA('WELLIE3.TSOBIN.DOC')


 INMR901I Dataset RCONWAY.POSIX.TSOCMD.DOC from RCONWAY on WTSCMXA
 INMR906A Enter restore parameters or 'DELETE' or 'END' +
```

Follow the instructions in the unloaded TSOCMD.DOC data set for instructions on how to further install the tso shell command.

The PRECOMP and HDRCPY REXX EXECs can be copied to a REXX or CLIST library that is allocated to your SYSEXEC or SYSPROC concatenation. Speak to your MVS system programmer if you need assistance.

The POLL.C and DB2CPGM2.I data sets can be copied to a directory in the HFS using the TSO/E OPUT command.

Appendix C. OpenEdition MVS Shell Command to Run TSO Commands

If you are logged on to TSO/E before you enter the OpenEdition MVS shell, you have the ability to switch back to TSO/E to execute TSO/E commands or REXX EXECs by pressing the TSO function key. When you switch to TSO/E, your shell work is suspended until you press the PA1 key to switch back to the shell. For more information, see *OpenEdition MVS User's Guide*. To eliminate this switching, a local shell command called `tso` was developed to enable the shell user to issue certain TSO/E commands directly from the shell and from shell scripts. This command was used during this port and is shown in some of the examples in this book.

The `tso` command is packaged in three pieces:

- tso - this is the main module. It resides in the HFS.
- tsopp - this is the postprocessor. It resides in the HFS.
- BPXWDYN - this is called to perform dynamic allocations. It resides in a MVS load module data set that you can either add to your system LINKLIST concatenation (member LNKLSTxx in SYS1.PARMLIB) or allocate to a JOBLIB or STEPLIB ddcard for each user.

For further usage information, see the documentation that is packaged with the command on the diskette.

Appendix D. Staying in Running Mode

This code allows a user who enters the shell from TSO/E to remain in *RUNNING* mode indefinitely. This can be a significant usability enhancement but it may have a performance impact if used by many users. The code is supplied on an “as is” basis.

```
/* poll program to keep the shell from dropping into INPUT mode  */
 #include <stdio.h>
 #include <signal.h>
 #include <unistd.h>

 void catcher(int signum) {
   exit(42);
 }

 main() {
   struct sigaction sa;
   char buf[1] = {0x2f};

   sigemptyset(&sa.sa_mask);
   sa.sa_flags = 0;
   sa.sa_handler = catcher;
   sigaction(SIGTERM, &sa, NULL);

   do {
     write(STDOUT_FILENO, buf, 1);
     sleep(4);
   } while (getppid() != 1);
 }
```

The supplied code, poll.c should be compiled and linked using c89 and the executable called out from the user’s *$HOME/.profile* as a background task, for example:

```
 echo "executing poll now"
 poll &
```

Appendix E. Copying the C Header Files to the HFS

The following REXX EXEC will copy the C header files into a directory in the HFS for use with utilities like grep.

```
/* REXX EXEC  */
/*  Copy the C header files from a MVS data set into the HFS        */
/*                                                                  */
/*  Note:  The C header files exist as ALIAS entries in the         */
/*         EDC.SEDCDHDR and OMVS.SFOMHDRS data sets.  The alias     */
/*         names are the common UNIX type names, so we only copy    */
/*         the aliases.  This exec works for those header files     */
/*         that are packaged with MVS/ESA SP 5.1 and AD/CYCLE       */
/*         C/370 1.2 only.                                          */
/*                                                                  */
/*  See HELP (below) for syntax of this command.                    */
/*                                                                  */
lc = 'abcdefghijklmnopqrstuvwxyz_'
uc = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ@'
ARG pdsin hfsout                              /* Supply input and outp.*/
if pdsin = '' | hfsout = '' | pdsin = '?',    /* Need help?    */
   | pdsin = 'HELP'  then call HELP
pdsin  = strip(pdsin,,"'")                    /* remove quotes, if any */
hfsout = strip(hfsout,,"'")                   /* remove quotes, if any  */
hfsout = translate(hfsout,lc,uc)              /* make it lowercase     */
  do
   L = length(hfsout)
   if substr(hfsout,1,1) <> '/' | substr(hfsout,L,1) <> '/'  then
     do
     say 'Target path does not start and end with a / '
     say 'Please verify and re-run HDRCPY.'
     exit
     end
  end
call CHECK_ARGS                               /* Check datasets.       */
x = OUTTRAP(member.)                          /* "Trap" output.        */
"LISTD (""pdsin"") MEMBERS"                   /* Get list of members.  */
x = OUTTRAP(OFF)                              /* Stop "trapping" output*/
i = 1                                         /* Initialize LISTD cnt. */
 do while i <= member.0                       /* Step thru LISTD list. */
   if SUBSTR(member.i,3,6) = MEMBER then do   /* Find word 'MEMBER'.   */
      z = i+1                                 /* Line after 'MEMBER'.  */
      i = member.0 + 1                        /* Quit DO WHILE.        */
      outcnt = 1                              /* Set output counter.   */
      do while z <= member.0                  /* Step thru members.    */
         memb = SUBSTR(member.z,3,LENGTH(member.z)-2)
         call MAKE_OUTPUT                     /* OPUTX the results.    */
         z = z + 1                            /* Increment member cnt. */
      end                                     /* END for DO WHILE Z..  */
   end                                        /* END for THEN DO.      */
   else i = i+1                               /* Increment LISTD cnt.  */
 end                                          /* END for DO WHILE I..  */
exit                                           /* Done!                 */

HELP:                                          /* Give help.            */
say '*************************************************************'
say
say 'HDRCPY will copy the C header files from your            '
```

```
say '       SFOMHDRS and SEDCDHDR data sets into the HFS.'
say
say 'The correct syntax of HDRCPY is:'
say
say 'HDRCPY <PDSin> <hfsout> '
say
say 'where:'
say '    PDSin = fully qualified name of the source data set, '
say '            (either SFOMHDRS or SEDCDHDR).'
say '    hfsout = pathname of the target HFS directory,    '
say '             where you want to copy the header files.    '
say
say '    Note: If all 2 are not supplied, HELP will be supplied.'
say
say 'Example 1: To copy the header files in dataset EDC.SEDCDHDR,       '
say 'into the HFS directory   /u/wellie2/oehead/   use the command:'
say
say '        HDRCPY EDC.SEDCDHDR /u/wellie2/oehead/  '
say
say 'Example 2: To copy the header files in dataset OMVS.SFOMHDRS,      '
say 'into the HFS directory   /u/wellie2/oehead/   use the command:'
say
say '        HDRCPY OMVS.SFOMHDRS /u/wellie2/oehead/  '
say '*************************************************************'
exit
return

CHECK_ARGS:                                 /* Verify before starting*/
x = LISTDSI(""'pdsin"'")                    /* Check input dataset.  */
if SYSREASON ¬=0 then do                    /* LISTDSI not good.     */
   say 'Your input dataset, 'pdsin', is not valid.' /* Explain.      */
   say 'REASON CODE return from LISTDSI function is:' SYSREASON
   say 'Please verify and re-run HDRCPY.'
   exit
end
else do
if syscalls('ON')>3 then                    /* Establish OE Rexx env */
do
say 'Unable to establish the SYSCALL environment'
return
end
address syscall
  "readdir (hfsout) dir."                  /* Check output directory*/
   if retval ¬=0 then do                     /* readdir failed     */
      say 'Your output directory, 'hfsout', is not valid or '
      say 'does not exist.  The values returned from the readdir function are:'
      say 'retval =' retval
      say 'errno =' errno
      say 'errnojr =' errnojr
      say 'Please verify and re-run HDRCPY.'
    exit
   end
end
return;

MAKE_OUTPUT:
  parse var memb var1 '(' var2 +1 var3 ')'  /*We copy the alias name ONLY*/
  filen = var3                              /* filename  = MEMBERNAME */
  filen = translate(filen,lc,uc)            /*make file name lower case*/
```

```
  if var3 = ''  then return                  /*Member with no alias - discard */
  else
 "oputx '"pdsin"("var3")' "hfsout""filen" suffix(h)" /*Perform OPUTX */
                                                     /* with .h suffix */
return
```

Appendix F. Modified Version of DB2 IVP Program for Testing CAF

This is a listing of the modified program, including the modifications necessary for using CAF. See 7.4.3, “Example of Executing a DB2/MVS Program Under OpenEdition MVS” on page 60, for the instructions on getting this code to run in the OpenEdition MVS environment.

You may need to change the following:

1. Under /*General declarations*/, set #define OMVS 1 if running in the OpenEdition MVS environment.
2. Under /*Input/Output files*/, the file names of the cardin and report files. They are currently set to cardin and report for running in the OpenEdition MVS environment.
3. Under /*CAF Start*/, set the name of the DB2 plan and the name of the local DB2/MVS subsystem.

```
/*********************************************************************/
/** MVS & OpenEdition MVS dependencies  see line 18 ********/
/***DB2 dependencies  see lines 182 and 187 ****************/
/*********************************************************************/
/* Include external definitions                                      */
/*********************************************************************/
#pragma linkage(dsnali, OS)
#pragma linkage(dsntiar, OS)
/*********************************************************************/
/* Include C library definitions                                     */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*********************************************************************/
/* General declarations                                              */
/*********************************************************************/
#define OMVS     1 /*0 for MVS batch, 1 for OpenEdition MVS */
#define NOTFOUND 100
/*********************************************************************/
/* Input / Output files                                              */
/*********************************************************************/
FILE *cardin;         /* Input control cards              */
FILE *report;         /* Output phone report              */
/*********************************************************************/
/* Input record structure                                            */
/*********************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
struct {
  char action[2];            /* L for list or U for update  */
  char lname[16];            /* last name or pattern- L mode*/
  char fname[13];            /* first name or pattern-L mode*/
  char eno[7];               /* employee number- U mode     */
  char newno[5];             /* new phone number- U mode    */
       } ioarea;
char trail[43];              /* unused portion of input rec */
char slname[16];             /* unmodified last name pattern*/
EXEC SQL END DECLARE SECTION;
/*********************************************************************/
/* Report headings                                                   */
```

```
/*********************************************************************/
struct {
  char hdr011[30];
  char hdr012[32];
       } hdr0 = {
                       "    REQUEST  LAST NAME        ",
                       "FIRST NAME    EMPNO   NEW XT.NO"};
#define rpthdr0 hdr0.hdr011
struct {
  char hdr111[29];
  char hdr112[21];
  char hdr113[30];
       } hdr1 = {
                       "-----------------------------",
                       " TELEPHONE DIRECTORY ",
                       "-----------------------------"};
#define rpthdr1 hdr1.hdr111
struct {
  char hdr211[10];
  char hdr212[11];
  char hdr213[ 8];
  char hdr214[ 6];
  char hdr215[ 9];
  char hdr216[ 5];
  char hdr217[ 5];
  char hdr221[ 7];
  char hdr222[ 7];
  char hdr223[ 5];
  char hdr224[ 5];
  char hdr225[ 5];
       } hdr2 = {
                       "LAST NAME",
                       "FIRST NAME",
                       "INITIAL",
                       "PHONE",
                       "EMPLOYEE",
                       "WORK",
                       "WORK",
                       "NUMBER",
                       "NUMBER",
                       "DEPT",
                       "DEPT",
                       "NAME"
                 };
#define rpthdr2 hdr2.hdr211,hdr2.hdr212,hdr2.hdr213,hdr2.hdr214,\
                hdr2.hdr215,hdr2.hdr216,hdr2.hdr217,hdr2.hdr221,\
                hdr2.hdr222,hdr2.hdr223,hdr2.hdr224,hdr2.hdr225
/*********************************************************************/
/* Report formats                                                    */
/*********************************************************************/
static char fmt1[] = "\n %s\n";
static char fmt2[] = " %9s%17s%10s%6s%10s%5s%5s\n%43s%7s%7s%5s%5s\n";
static char fmt3[] = " %-16s%-16s%-5s%-7s%-9s%-5s%-36s\n";
static char fmt4[] = " %1c%15c%12c%6c%4c%43c";
static char fmt5[] =
   "\n\n %s\n %s\n   --%-7s--%-15s--%-12s--%-5s--%-9s--\n";
/*********************************************************************/
/* Fields sent to message routine                                    */
/*********************************************************************/
```

```
char outmsg[70];             /* error/information msg buffer*/
char module[ 8] = "DSN8BD3"; /* module name for message rtn */
/* extern DSN8MDG(); *?               /* message routine            */
/********************************************************************/
/* SQL communication area                                           */
/********************************************************************/
EXEC SQL INCLUDE SQLCA;
/********************************************************************/
/* SQL declaration for view VPHONE                                  */
/********************************************************************/
EXEC SQL DECLARE VPHONE TABLE
       (LASTNAME       VARCHAR(15)  NOT NULL,
        FIRSTNAME      VARCHAR(12)  NOT NULL,
        MIDDLEINITIAL     CHAR( 1)  NOT NULL,
        PHONENUMBER       CHAR( 4)          ,
        EMPLOYEENUMBER    CHAR( 6)  NOT NULL,
        DEPTNUMBER        CHAR( 3)  NOT NULL,
        DEPTNAME       VARCHAR(36)  NOT NULL);
/********************************************************************/
/* Structure for pphone record                                      */
/********************************************************************/
/* Note: since the sample program data does not contain imbedded    */
/*       nulls, the C language null terminated string can be used to */
/*       receive the varchar fields from DB2.                       */
EXEC SQL BEGIN DECLARE SECTION;
struct {
  char lastname[16];
  char firstname[13];
  char middleinitial[2];
  char phonenumber[5];
  char employeenumber[7];
  char deptnumber[4];
  char deptname[37];
  } pphone;
EXEC SQL END   DECLARE SECTION;
/********************************************************************/
/* SQL declaration for view VEMPLP (used for update processing)     */
/********************************************************************/
EXEC SQL DECLARE VEMPLP TABLE
       (EMPLOYEENUMBER    CHAR( 6)  NOT NULL,
        PHONENUMBER       CHAR( 4)          );
/********************************************************************/
/* Structure for pemplp record                                      */
/********************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
struct {
  char employeenumber[7];
  char phonenumber[5];
       } pemplp;
EXEC SQL END   DECLARE SECTION;
/********************************************************************/
/* SQL cursors                                                      */
/********************************************************************/
/* cursor to list all employee names */
EXEC SQL DECLARE TELE1 CURSOR FOR
         SELECT *
         FROM VPHONE;
/* cursor to list all employee names with a pattern */
```

```
/* (%) or (_) in last name                          */
EXEC SQL DECLARE TELE2 CURSOR FOR
         SELECT *
         FROM VPHONE
         WHERE LASTNAME LIKE :lname;
/* cursor to list all employees with a specific last name */
EXEC SQL DECLARE TELE3 CURSOR FOR
         SELECT *
         FROM VPHONE
         WHERE LASTNAME = :slname
           AND FIRSTNAME LIKE :fname;
/*********************************************************************/
/* SQL return code handling                                          */
/*********************************************************************/
EXEC SQL WHENEVER SQLERROR   GOTO DBERROR;
EXEC SQL WHENEVER SQLWARNING GOTO DBERROR;
EXEC SQL WHENEVER NOT FOUND  CONTINUE;
/* Start main routine********************************************** */
extern main()
{
/* ***CAF Start *** ************************************************ */
long int fnret;
char     opnfunc[13] = "OPEN ";
char     clsfunc[13] = "CLOSE ";
char     plan[9] = "DB2CPLAN ";  /*name of DB2 PLAN*/
char     term_opt[5] = "SYNC";
long int return_code;
long int reason_code;
/* extern char     DB2SSID[]; */
char     DB2SSID[4] = "DB3B";   /*name of DB2 subsystem*/
char     result_msg[64] = "***result_msg***";
int      *rpc_fault_status;
int      r_f_s;
/* ***CAF End*** *************************************************** */
/* ***CAF Start Open the plans connections with DB2***************** */
printf("main start\n");
rpc_fault_status = &r_f_s;
fnret = dsnali(opnfunc,
               DB2SSID,
               plan,
               &return_code,
               &reason_code);
printf("Open request fnret = %d\n", fnret);
printf("Open request return_code = %d\n", return_code);
printf("Open request reason_code = %d\n", reason_code);
fflush(NULL);
if (fnret != 0)
{
 sprintf(result_msg,
         "*** Unable to open DB2 for return: *",
         "%ld REASON: %ld",
         return_code,
         reason_code);
 *rpc_fault_status = 1;
 printf("main end\n");
 return;
}
/* ***CAF End****************************************************** */
  /* Open the input and output files */
```

```
  if (OMVS == 0)
      cardin = fopen("DD:CARDIN","r");
      else
      cardin = fopen("cardin","r");
  if (cardin == NULL)
      {
       printf("Open error cardin\n");
       exit(16);
      }
  if (OMVS == 0)
      report = fopen("DD:REPORT","w");
      else
      report = fopen("report","w");
  if (report == NULL)
      {
       printf("Open error report\n");
       exit(16);
      }
  /* While more input, process */
  while (!feof(cardin))
  {
    /* Read the next request */
    if (fscanf(cardin, fmt4,
                          ioarea.action,
                          ioarea.lname,
                          ioarea.fname,
                          ioarea.eno,
                          ioarea.newno,
                          trail) == 6)
    {
      /* Display the request */
      DSN8MDG(module, "000I", outmsg);
      fprintf(report, fmt5,
                            outmsg,
                            rpthdr0,
                            ioarea.action,
                            ioarea.lname,
                            ioarea.fname,
                            ioarea.eno,
                            ioarea.newno);
      Do_req();
    }
  } /* endwhile */
  fclose(report);
/* ***CAF Start Close the plans connections with DB2**************** */
fnret = dsnali(clsfunc,
               term_opt,
               &return_code,
               &reason_code);
printf("Close request fnret = %d\n", fnret);
printf("Close request return_code = %d\n", return_code);
printf("Close request reason_code = %d\n", reason_code);
fflush(NULL);
if (fnret != 0)
{
 sprintf(result_msg,
         "*** Unable to close DB2 for return: *",
         "%ld REASON: %ld",
         return_code,
```

```
        reason_code);
 *rpc_fault_status = 1;
 printf("main end\n");
 return;
}
/* ***CAF End*** ************************************************** */
 printf("main end\n");
} /* end main */
/*********************************************************************/
/* Process the current request                                       */
/*********************************************************************/
Do_req()
{
  char *blankloc;                      /* string translation pointer  */
  strcpy(slname, ioarea.lname);        /* save untranslated last name */
  while (blankloc = strpbrk(ioarea.lname, " "))
    *blankloc = '%';                   /* translate blanks into %     */
  while (blankloc = strpbrk(ioarea.fname, " "))
    *blankloc = '%';                   /* translate blanks into %     */
  /* Determine request type */
  switch (ioarea.action[0])
  {
    /* Process LIST request */
    case 'L':
      /* Print the report headings */
      fprintf(report, fmt1, rpthdr1);
      fprintf(report, fmt2, rpthdr2);
      /* List all employees */
      if (!strcmp(slname,"*             ")){
        EXEC SQL OPEN TELE1;
        EXEC SQL FETCH TELE1 INTO :pphone;
        if (sqlca.sqlcode == NOTFOUND){          /* If no employees   */
          DSN8MDG(module, "008I", outmsg);       /* found, display    */
          fprintf(report, " %s\n", outmsg);      /* error message     */
        } /* endif */
        while (sqlca.sqlcode == 0){
          Prt_row();
          EXEC SQL FETCH TELE1 INTO :pphone;
        } /* endwhile */
        EXEC SQL CLOSE TELE1;
        /* List generic employees */
      } else {
        if (strpbrk(slname, "%")) {
          EXEC SQL OPEN TELE2;
          EXEC SQL FETCH TELE2 INTO :pphone;
          if (sqlca.sqlcode == NOTFOUND){        /* If no employees   */
            DSN8MDG(module, "008I", outmsg);     /* found, display    */
            fprintf(report, " %s\n", outmsg);    /* error message     */
          } else {
            while (sqlca.sqlcode == 0){
              Prt_row();
              EXEC SQL FETCH TELE2 INTO :pphone;
            } /* endwhile */
          } /* endif */
          EXEC SQL CLOSE TELE2;
          /* List specific employee */
        } else {
          EXEC SQL OPEN TELE3;
          EXEC SQL FETCH TELE3 INTO :pphone;
```

```
          if (sqlca.sqlcode == NOTFOUND){        /* If no employee    */
            DSN8MDG(module, "008I", outmsg);     /* found, display    */
            fprintf(report, " %s\n", outmsg);    /* error message     */
          } else {
            while (sqlca.sqlcode == 0){
              Prt_row();
              EXEC SQL FETCH TELE3 INTO :pphone;
            } /* endwhile */
          } /* endif */
          EXEC SQL CLOSE TELE3;
        } /* endif */
      } /* endif */
      break; /* end of 'L' request */
    /* Update an employee phone number */
    case 'U':
      EXEC SQL UPDATE VEMPLP
               SET PHONENUMBER = :ioarea.newno
               WHERE EMPLOYEENUMBER = :ioarea.eno;
      if (sqlca.sqlcode == 0){                   /* If employee       */
        DSN8MDG(module, "004I", outmsg);         /* updated, display  */
        fprintf(report, " %s\n", outmsg);        /* confirmation msg  */
      } else {
        DSN8MDG(module, "007E", outmsg);         /* otherwise, display*/
        fprintf(report, " %s\n", outmsg);        /* error message     */
      } /* endif */
      break;
    /* Invalid request type */
    default:
      DSN8MDG(module, "068E", outmsg);           /* Display error msg */
      fprintf(report, " %s\n", outmsg);
  } /* endswitch */
  return;
DBERROR:
  Sql_err();
} /* end Do_req */
/*********************************************************************/
/* Print a single employee on the report                             */
/*********************************************************************/
Prt_row()
{
  fprintf(report, fmt3, pphone.lastname,
                        pphone.firstname,
                        pphone.middleinitial,
                        pphone.phonenumber,
                        pphone.employeenumber,
                        pphone.deptnumber,
                        pphone.deptname);
}
/*********************************************************************/
/* SQL error handler                                                 */
/*********************************************************************/
Sql_err() {
#define data_len 120
#define data_dim 8
struct error_struct {
  short int error_len;
  char error_text[data_dim][data_len];
  } error_message = {data_dim * data_len};
extern short int dsntiar(struct sqlca       *sqlca,
```

```
                        struct error_struct *msg,
                        int                 *len);
short int rc;
int i;
static int lrecl = data_len;
  DSN8MDG(module, "060E", outmsg);
  fprintf(report, " %s %i\n", outmsg, sqlca.sqlcode);
  rc = dsntiar(&sqlca, &error_message, &lrecl);  /* Format the sqlca  */
  if (rc == 0){                                  /* Print formatted   */
    for (i=0;i<=7;i++){                          /*   sqlca           */
      fprintf(report, "%.120s\n", error_message.error_text  [i]);
    } /* endfor */
  } else {
    DSN8MDG(module, "075E", outmsg);
    fprintf(report, " %s %hi\n", outmsg, rc);
  } /* endif */
  /* Attempt to rollback any work already done */
  EXEC SQL WHENEVER SQLERROR   CONTINUE;
  EXEC SQL WHENEVER SQLWARNING CONTINUE;
  EXEC SQL WHENEVER NOT FOUND  CONTINUE;
  EXEC SQL ROLLBACK;
  if (sqlca.sqlcode == 0){                       /* If rollback       */
    DSN8MDG(module, "053I", outmsg);             /* completed, display*/
    fprintf(report, " %s\n", outmsg);            /* confirmation msg  */
  } else {                                       /* otherwise, display*/
    DSN8MDG(module, "061E", outmsg);             /* error message     */
    fprintf(report, " %s %i\n", outmsg, sqlca.sqlcode);
  } /* endif */
  fclose(report);
  exit(0);
} /* end of Sql_err */
/* End main routine********************************************** */
/* Start error routine******************************************** */
DSN8MDG(module, icode, outmsg)
char outmsg[70];
char module[8];
char icode[5];
{
/*********************************************************************/
/* Miscellaneous declarations                                        */
/*********************************************************************/
#define NumMsgs 9
register int i;
char mnf[51] = "MESSAGE TEXT NOT FOUND                ";
/*********************************************************************/
/* Message code array                                                */
/*********************************************************************/
char code[NumMsgs][5] = {
          "000I",    "004I",    "007E",    "008I",
          "053I",    "060E",    "061E",
          "068E",    "075E"};
/*********************************************************************/
/* Message text array                                                */
/*********************************************************************/
char text[NumMsgs][51] = {
"REQUEST IS:"                                  ,/*000I*/
"EMPLOYEE SUCCESSFULLY UPDATED"                ,/*004I*/
"EMPLOYEE DOES NOT EXIST, UPDATE NOT DONE"     ,/*007E*/
"NO EMPLOYEE FOUND IN TABLE"                   ,/*008I*/
```

```
                                    /* GENERAL INFO. MESSAGES */
"ROLLBACK SUCCESSFUL, ALL UPDATES REMOVED"          ,/*053I*/
                                    /* GENERAL ERROR MESSAGES */
"SQL ERROR, RETURN CODE IS:"                        ,/*060E*/
"ROLLBACK FAILED, RETURN CODE IS:"                  ,/*061E*/
"INVALID REQUEST, SHOULD BE 'L' OR 'U'"             ,/*068E*/
"MESSAGE FORMAT ROUTINE ERROR, RETURN CODE IS:"     }/*075E*/;
/*********************************************************************/
/* Main program routine                                              */
/*********************************************************************/
/* Scan message table (0 thru NumMsgs - 1) for requested message */
for (i=0; (i<NumMsgs) && !(strcmp(icode,code[i])==0); i++){}
/* Found message code */
if (i<NumMsgs){
  strcpy(outmsg, "DSN8");
  strcat(outmsg, icode);
  strcat(outmsg, "   ");
  strcat(outmsg, module);
  strcat(outmsg, "-");
  strcat(outmsg, text[i]);
/* Unknown message code */
} else {
  strcpy(outmsg, "DSN8");
  strcat(outmsg, icode);
  strcat(outmsg, "   ");
  strcat(outmsg, module);
  strcat(outmsg, "-");
  strcat(outmsg, mnf);
} /* endif */
return;                             /* return to caller          */
} /* dsn8mpg */
/* End error routine*** ******************************************** */
```

Appendix G. REXX EXEC to Run the DB2 Precompiler and BIND from the OpenEdition MVS Shell

Currently, there is no built in shell command to run the DB2/MVS precompiler and DB2 DSN BIND process directly from the shell. It is possible to execute them as commands in the TSO environment, but it requires some knowledge of the DB2/MVS command syntax.

To make this process easier, a REXX EXEC was developed where the OpenEdition MVS user specifies, as parameters, the HFS file the DB2/MVS precompiler should use as input, and the HFS file were the processed C source should be placed as output. The name of the plan for the DB2 DSN bind process is also given as a parameter if required.

After having executed the REXX EXEC, the C source output was used as input to c89 for making the executable program under OpenEdition MVS. See 7.4.3, “Example of Executing a DB2/MVS Program Under OpenEdition MVS” on page 60, for a step-by-step procedure on how to prepare and run a DB2 program under OpenEdition MVS.

A listing of the REXX EXEC is shown below. It will be necessary to customize this EXEC to match your particular installation’s naming conventions for the DB2/MVS data sets and DB2 system names.

```
/* ***REXX Start********************************************************* */
/* Exec Name:    PRECOMP                                                  */
/*                                                                        */
/* Description:  Runs the DB2 precompiler.  This exec will copy           */
/*               C source from the hierarchical file system               */
/*               and place it into a temporary MVS data set for           */
/*               processing by the DB2 precompiler.  The output           */
/*               from the DB2 precompiler (modified C source) will        */
/*               be copied back into the hierarchical file system.        */
/*                                                                        */
/*               Optionally run BIND to bind to a plan.                   */
/*                                                                        */
/*               This exec can be run from the OpenEdition MVS shell.     */
/*                                                                        */
/* Invocation:   DSNHPC, BIND                                             */
/*                                                                        */
/* Variables :   DB2 subsystem name                                       */
/*               High level qualifier of DB2 data sets                    */
/*                                                                        */
/* Input parms:                                                           */
/*                                                                        */
/* in=            (required)  HFS INPUT FILE TO DB2 PRECOMPILER           */
/* ou=            (required)  HFS OUTPUT FILE FROM DB2 PRECOMPILER        */
/*                                                                        */
/* pa=            (optional)  PATH FOR BOTH INPUT and OUTPUT              */
/* pl=            (optional)  NAME OF PLAN IF YOU WANT TO BIND            */
/*                                                                        */
/* Examples:                                                              */
/*Run DB2 precompile but don’t bind it to a plan                          */
/*   precomp pa=/u/user/ in=db2cpgm2.i ou=db2cpgm2.c                      */
/*                                                                        */
/*Run DB2 precompile and bind it to a plan called db2cplan                */
```

```
/*   precomp pa=/u/user/ in=db2cpgm2.i ou=db2cpgm2.c                  */
/*           pl=db2cplan                                              */
/*                                                                    */
/*Run DB2 precompile & bind and have different input and output paths */
/*   precomp in=/u/user/db2cpgm2.i ou=/u/kjellaa/db2cpgm2.c           */
/*           pl=db2cplan                                              */
/*                                                                    */
/*                                                                    */
/* ***SET TSO ENVIRONMENT********************************************* */
ADDRESS TSO
/* ***FETCH REXX PARAMETER LIST*************************************** */
parse arg p.1  p.2  p.3  p.4  p.5
say 'PRECOMP started'
/* ***free and alloc sysprint and systerm***************************** */
'free  ddname(sysprint)'
'alloc ddname(sysprint) da(*)'
'free  ddname(systerm)'
'alloc ddname(systerm) da(*)'
/* ***Pick up TSO userid********************************************** */
uid = sysvar(SYSUID)                              /* ***TSO UID  *** */
/*                                                                    */
/* *** Define SYSTEM variables ************************************** */
SY = 'SYSTEM(DB3B)'        /*Name of DB2 subsystem*/
db2hlq = 'DSN310'          /*High level qualifier of DB2 data sets*/
/*                                                                    */
/* ***Define data set names****************************************** */
syslibd = ''''||db2hlq'.SRCLIB.DATA'||''''         /*****DB2********* */
dbrm    = ''''||db2hlq'.DBRMLIB.DATA'||''''        /* ***PLAN DSN *** */
preinp  = ''''||UID'.DB2.PREINP'||''''             /* ***WORK FILE*** */
preout  = ''''||UID'.DB2.PREOUT'||''''             /* ***WORK FILE*** */
/* ***Define REXX variables****************************************** */
pathi = ''
patho = ''
pathx = ''
plan  = ''
rch   = 0
rcx   = 0
/* ***CHECK P.1 THRU P.5 ********************************************* */
do i = 1 TO 5 BY 1
   if p.i <> '' then
      do
      say 'p.'||i||'='||p.i
      xxx = substr(p.i,1,3)
      p.i = xxx||substr(p.i,4)
      if substr(p.i,1,3) = 'pl=' then
         plan  = substr(p.i,4); else
      if substr(p.i,1,3) = 'in=' then
         pathi = substr(p.i,4); else
      if substr(p.i,1,3) = 'ou=' then
         patho = substr(p.i,4); else
      if substr(p.i,1,3) = 'pa=' then
         pathx = substr(p.i,4); else
         do
         say 'INVALID VALUE. PARAMETER #'||i||'='||p.i
         say 'PRECOMP ended with RC = 16'
         return 16
         end
      end
end
```

```
/* ***CONVERT TO UPPER CASE************************************** */
UPPER PLAN
/* ***TEST INPUT PARAMETER COMBINATIONS***************************** */
  if (pathi = '' |  patho = '')  then
   do
   say 'INPUT FILENAME, OUTPUT FILENAME NOT SPECIFIED'
   say 'PRECOMP ended with RC = 16'
   return 16
   end
if pathx <> '' & (pathi = '' | pathi = '')  then
   do
   say 'PATH SPECIFIED, BUT INPUT AND OUTPUT FILE NOT SPECIFIED'
   say 'PRECOMP ended with RC = 16'
   return 16
   end
if pathx <> ''  then
   do
   L = length(pathx)
   if substr(pathx,1,1) <> '/' | substr(pathx,L,1) <> '/'  then
      do
      say 'PATH SPECIFIED, BUT DOES NOT START AND END WITH /'
      say 'PRECOMP ended with RC = 16'
      return 16
      end
   end
if pathx <> '' & (pos('/',pathi) <> 0 | pos('/',patho) <> 0) then
   do
   say 'PATH SPECIFIED, BUT INPUT OR OUTPUT FILE NAME CONTAINS /'
   say 'PRECOMP ended with RC = 16'
   return 16
   end
if (pathi =  patho) then
   do
   say 'PRECOMPILE, BUT SAME PATH NAME FOR INPUT AND OUTPUT'
   say 'PRECOMP ended with RC = 16'
   return 16
   end
if plan <> ''  then
   do
   plen = length(plan)
      if plen > 8 then
      do
      say 'PLAN NAME IS TOO LONG, MUST BE 8 CHARACTERS OR LESS'
      say 'PRECOMP ended with RC = 16'
      return 16
      end
   end
/* ***CREATE SOME VARIABLES************************************** */
if pathx <> ''  then
   pathi = pathx||pathi
if pathi <> ''  then
   do
   pathi = ''''||pathi||''''                        /* ***HFS INP  *** */
   say 'pathi='||pathi
   end
if pathx <> ''  then
   patho = pathx||patho
if patho <> ''  then
   do
```

```
   patho = ''''||patho||''''                          /* ***HFS OUT  *** */
   say 'patho='||patho
   end
/* ***DELETE AND ALLOCATE MVS DATA SETS***************************** */
   do
   if sysdsn(preinp) = 'OK' then
      "delete ("preinp") nonvsam"
   if sysdsn(preout) = 'OK' then
      "delete ("preout") nonvsam"
   end
/* ***ALLOCATE WORK FILES******************************************** */
   do
     if sysdsn(preinp) <> 'OK' then
        'alloc fi(preinp) da('preinp') unit(sysallda) new reuse,
         space(16,16) tracks recfm(f b) lrecl(80)'; else
        'alloc fi(preinp) da('preinp') recfm(f b) lrecl(80) shr reuse'
     if sysdsn(preout) <> 'OK' then
       'alloc fi(preout) da('preout') unit(sysallda) new reuse,
        space(16,16) tracks recfm(f b) lrecl(80)'; else
       'alloc fi(preout) da('preout') recfm(f b) lrecl(80) shr reuse'
    end
/* ***ALLOCATE HFS FILES******************************************** */
   do
    'allocate ddname(hfsinp) path('pathi') pathopts(ordwr,ocreat)',
             'pathmode(sirusr,siwusr)'
   end
/* ***OCOPY FROM OMVS TO MVS**************************************** */
say 'OCOPY C input source from HFS'
   do
   say 'OCOPY DB2 PRECOMPILE'
   'ocopy indd(hfsinp) outdd(preinp) text convert(no)'
   call setrch
   say 'OCOPY ended with RC =' rcx
   'free  ddname(preinp)'
   if rcx > 0 then
      return rcx
   end
/* ***FREE INPUT FILES********************************************** */
  'free ddname(hfsinp)'
/* ***executing the DB2 precompiler********************************* */
if plan = '' then
   do
   plan = 'DUMMY'       /*dummy name for plan to get through precomp  */
   end
   do
   dbrmm  = ''''||db2hlq'.DBRMLIB.DATA('PLAN')'||''''
   'alloc ddname(sysin)  da('preinp')    shr reuse'
   'alloc ddname(syscin)  da('preout')   shr reuse'
   'alloc ddname(syslib)  da('syslibd')  shr reuse'
   'alloc ddname(dbrmlib) da('dbrmm')    shr reuse'
   'alloc fi(sysut1)  unit(vio) new reuse space(16 16) track'
   'alloc fi(sysut2)  unit(vio) new reuse space(16 16) track'
   PROG = "'X.Y(DSNHPC)'"
   PARM = "'HOST(C),MARGINS(1,80),NOOPTIONS,FLAG(W)'"
   call pgmcall  prog parm
   call setrch
   say 'DB2 precompiler ended with RC =' rcx
   'free  ddname(sysin)'
   'free  ddname(syscin)'
```

```
   'free  ddname(syslib)'
   'free  ddname(dbrmlib)'
   if rcx > 4 then
      return rcx
end
/* ***executing the DB2 DSN BIND command***************************** */
if plan <> 'DUMMY' then
   do
  'alloc ddname(dbrmlib) da('dbrm') shr reuse'
   queue 'BIND PLAN('PLAN') MEMBER('PLAN') ACT(REP) ISOLATION(CS) FLAG(W)'
   queue 'END'
  'DSN' SY
   call setrch
   say 'DB2 DSN BIND ended with RC =' RC
  'free  ddname(dbrmlib)'
   if rcx > 0 then
      return rcx
   end
/* ***executing OCOPY to copy translated file to hfs***************** */
if patho <> ''  then
   do
      say 'OCOPY C translated source to HFS'
     'free  ddname(preout)'
     'alloc ddname(mvsinp) da('preout') shr reuse'
     'alloc ddname(hfsout) path('patho') pathopts(ordwr,ocreat)',
          'pathmode(sirusr,siwusr)'
     'ocopy indd(mvsinp) outdd(hfsout) text convert(no)'
      call setrch
      say 'OCOPY ended with RC =' rcx
     'free  ddname(mvsinp)'
     'free  ddname(hfsout)'
      if rcx > 0 then
         return rcx
   end
SAY 'PRECOMP ended with highest RC =' rch
RETURN RCH
/* ***pgmcall subroutine for calling a program with a parm field***** */
pgmcall:
parse arg pgm a
trace o
a = strip(a,"B","'")
p = length(a)
interpret "pl = '"d2x(length(a))"'x"
pl = overlay(pl,'0000'x,3-length(pl)) || a
parse var pgm "(" pg ")"
ADDRESS "LINKPGM" pg  "pl"
return RC
/* ***setrc subroutine for keeping highest return code*************** */
setrch:
rcx = rc
if rcx > rch then
   rch = rcx
return
/* ***REXX End******************************************************** */
```

Appendix H. MVS Terminology

3270. The protocol used by IBM mainframe systems for interactive terminal connection. The data sent to the screen is referred to as a 3270 datastream and includes the data to be displayed together with control characters to utilize specialized functions (for example, cursor movements, hidden input) of the 3270 devices.

CLIST. A stored list of TSO commands to be executed sequentially to perform a task. It is analogous to a shell script.

DASD. Direct Access Storage Devices is the IBM mainframe world's term for disks. They are connected together in "strings," which attach to DASD controllers. The controllers are attached to the mainframe by "channels."

Data Set. An MVS file. Or in the case of a partitioned data set (PDS) a collection of files. See HFS.

DFSMS. DFSMS is the family of software products that implement Systems Managed Storage (SMS). At full implementation, SMS automatically manages disk storage on an MVS system in accordance with user-defined policies. These policies cover such areas as performance, availability and backup. The MVS system can automatically determine where to place files to achieve optimum performance, move them elsewhere as devices become full, take backup copies whenever changes are made, migrate to tape if not used for a certain time and so on. The HFS data set must be "SMS-managed."

EBCDIC. Extended Binary-Coded Decimal Interchange Code (EBCDIC) is the hexadecimal character encoding scheme used by IBM mainframes and other IBM computers. Think of it as IBM's ASCII.

HFS. Prior to OpenEdition, MVS had never had a hierarchical filesystem. MVS calls its files, data sets. There are a number of different data set types. A "partitioned" data set actually contains many individual files called members but there is no hierarchy among the members. MVS data sets are defined with a number of attributes that describe how the file is organized. MVS keeps track of all its files by using a number of catalogs. OpenEdition added a new data set to MVS, the HFS data set. In UNIX terms each HFS data set is a mountable filesystem. Traditional MVS services can operate on HFS data sets without knowing or caring that they are really lots of different files. OpenEdition services can operate at the (UNIX) file level.

ISPF. ISPF and TSO are analogous to a UNIX shell. TSO is the interactive environment in which systems programmers (that is, systems administrators) and application developers work. ISPF is a set of front-end menus to raw TSO commands. It is used to create data sets (that is, files), copy, edit, browse, delete and so on. You will have to log on to TSO. You will probably want your MVS systems programmer to set up your TSO profile (he'll call it a proc) to put you into a shell automatically.

JCL. Job control language (JCL) is used to submit batch jobs to MVS. It is code that specifies such things as what files will be used, value of parameters to be passed, where output is to go, and so on.

JES2/3. There are two versions of the Job Entry Subsystem (JES). An installation will either have JES2 or JES3. The differences are mainly historical and of no concern. JES is the bit of MVS that reads the JCL and is responsible for getting batch jobs into the system and running with everything they need.

Logon/Logoff. In the IBM mainframe world, users log on and log off rather than log in and log out. The concept is identical except that a TSO user may only log on in one place at one time. It is not possible to have multiple instances of the same user on the system at the same time - either from one workstation or several.

LPAR mode. Mainframes that run in logically partitioned (LPAR) mode can be logically split up into what looks like a number of smaller mainframes (called partitions). The total available hardware resources (processing power, main memory and I/O channels) are shared among the partitions. Each partition then runs its own operating system and operates completely independent of the other partitions. In this way, new and test partitions can be provided without requiring a new mainframe. See also VM.

OpenEdition MVS. POSIX compliant MVS. MVS is IBM′s primary mainframe operating system. It is very strong in the areas of reliability, data integrity, security and so on. It is also able to handle much larger quantities of data and much higher numbers of concurrent users than traditional UNIX systems. OpenEdition MVS runs all existing MVS programs unaltered, but integrates a shell, a hierarchical filesystem and POSIX API. The IBM marketing position is that it combines the best of the two worlds. For UNIX developers it opens up a whole new market. OpenEdition MVS is a commercial UNIX.

RACF. Resource Access Control Facility (RACF) is MVS′s security product. It controls access to resources (usually files) in an MVS system. It is more flexible than the “user or group or everybody else” approach of permission bits. In OpenEdition, permission bits are used within each HFS, and RACF controls access to the HFS itself. RACF also controls access to the OpenEdition services. Thus, TSO users can only use the shell if they have specifically been given RACF authority to do so.

REXX. REXX is a command language similar to CLISTs. It is supported on AIX, OS/2 and other IBM systems.

RMF/SMF. Every time anything at all happens on an MVS system a Systems Management Facility (SMF) record is written about it. SMF is similar to sar in UNIX. SMF records provide the raw material for accounting and performance tools to report and analyze. Resource Measurement Facility (RMF) is one such performance reporting tool.

SDSF. System Display and Search Facility (SDSF) is an MVS product for handling job output. It can be used to view, print, redirect, delete output created by batch.

System/360, System/370, System/390 architecture. The architecture of IBM mainframes as it has evolved. Programs written to conform to the original S/360 architecture introduced in 1964 still run unchanged on today′s S/390 mainframes.

TSO. See ISPF

VM. Another IBM mainframe operating system. One of its strengths is its ability to run other operating systems, for example, as “guests” underneath it. This is particularly useful in a test environment. Thus, a small OpenEdition test or development environment can easily be established without requiring a separate mainframe on which to run. See also LPAR mode.

VSAM. In UNIX systems, the application is responsible for applying structure to a file. An ASCII file is a long string of bytes. Within MVS, *access methods* of which Virtual Storage Access Method (VSAM) is the most common, are part of the operating system, and they provide a structure (in terms of record length, indexes and so on) to files. Copying a VSAM file to the HFS (via a sequential data set) loses the structure.

Index

N

P

R

S

T

V

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
Porting Applications to the
OpenEdition MVS Platform
April 1995

Publication No. GG24-4473-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction	____		
Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

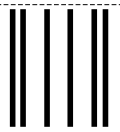
_____ _____
Name Address

_____ _____
Company or Organization

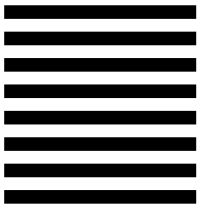
_____ _____
Phone No.

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 International Technical Support Organization
Mail Station P099
522 SOUTH ROAD
POUGHKEEPSIE NY
USA 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.