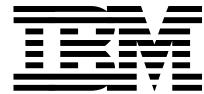
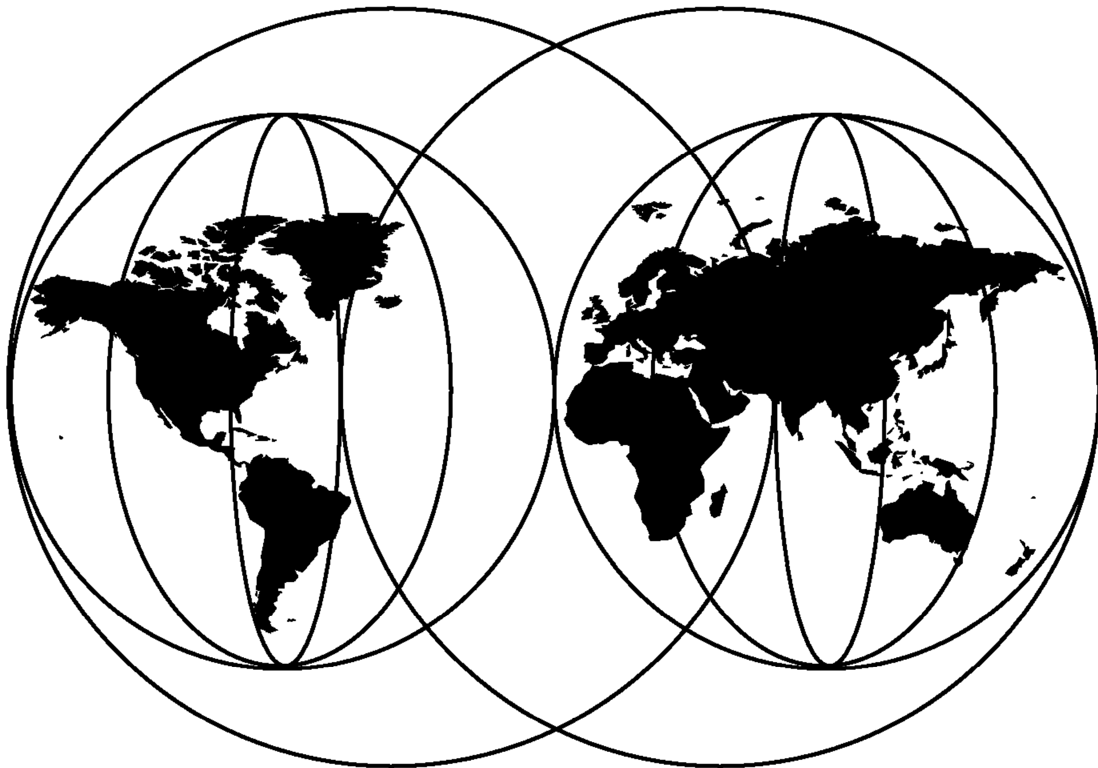


VM/ESA
Network Computing with Java and NetRexx

*Kris Buelens ** Bengt Heijnesson ** Dave Jones ** Salvador Torres*

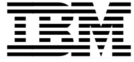


International Technical Support Organization

http://www.redbooks.ibm.com

This book was printed at 240 dpi (dots per inch). The final production redbook with the RED cover will be printed at 1200 dpi and will provide superior graphics resolution. Please see "How to Get ITSO Redbooks" at the back of this book for ordering instructions.

SG24-5148-00



International Technical Support Organization

SG24-5148-00

VM/ESA
Network Computing with Java and NetRexx

November 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special Notices” on page 161.

First Edition (November 1998)

This edition applies to Virtual Machine/Enterprise Systems Architecture (VM/ESA), Version 2 Release 3.0, Program Number 5654-030, and subsequent releases.

Note

This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ 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.

© **Copyright International Business Machines Corporation 1998. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Figures . . . ix

Tables . . . xi

Preface . . . xiii
The Team That Wrote This Redbook . . . xiii
Comments Welcome . . . xiv

Chapter 1. Introduction . . . 1

Chapter 2. Overview of NetRexx and Java on VM/ESA . . . 3
2.1 Java, NetRexx, OpenEdition, and the BFS . . . 3
2.2 Overview of the SFS . . . 3
2.2.1 SFS Servers and File Pools . . . 3
2.3 Overview of the BFS . . . 8
2.3.1 POSIX Terminology . . . 8
2.3.2 Directory Entries for POSIX BFS Usage . . . 10
2.4 Some Common SFS and BFS Commands . . . 11
2.5 The Java Environment Under VM . . . 12
2.6 SFS and BFS Directory Structures . . . 13
2.6.1 An SFS File Space . . . 13
2.6.2 A BFS File Space . . . 14
2.6.3 Combining File Spaces : SFS Aliases - BFS Links . . . 14
2.7 Installing Java and NetRexx without the Shell and Utilities . . . 16
2.7.1 Major Steps to Install “Shell-less” Java and NetRexx . . . 17
2.8 Adding a NetRexx Developer User ID . . . 20

Chapter 3. Tools Used During the Project . . . 21
3.1 XEDIT . . . 21
3.1.1 PROFILE XEDIT . . . 21
3.2 BFSLIST - Listing the Contents of a BFS Directory . . . 22
3.2.1 OPENVM LISTFILE . . . 22
3.2.2 POSIX Shell and Utilities . . . 23
3.2.3 BFSLIST . . . 24
3.3 BFSTREE - Listing a BFS Directory Tree . . . 27
3.4 NetRexx Compile . . . 29
3.4.1 NRC EXEC - NetRexx Compile . . . 29
3.4.2 NRC XEDIT - NetRexx Compile . . . 30
3.5 JC EXEC - Java Compile . . . 31
3.6 NetRexx Run . . . 31
3.6.1 NRR EXEC - NetRexx Run . . . 31
3.6.2 NRR XEDIT - NetRexx Run . . . 32
3.7 Tools for the POSIX Shell Users . . . 32
3.8 SETCENV - Setting C Environment Variables . . . 33
3.8.1 Important Environment Variables . . . 33
3.8.2 Setting Environment Variables from CMS . . . 34
3.8.3 More About Classpath . . . 34

Chapter 4. Comparing REXX to NetRexx . . . 37
4.1 REXX′s Position . . . 37
4.1.1 The REXX Language . . . 37

© Copyright IBM Corp. 1998

4.1.2 REXX Compilers . . . 38
4.1.3 Hello World in REXX . . . 38
4.2 NetRexx's Position . . . 38
4.2.1 Hello World in NetRexx . . . 38
4.2.2 Hello World in Java . . . 38
4.2.3 The NetRexx Language . . . 39
4.2.4 NetRexx and Compilers . . . 39
4.3 NetRexx Syntax Introduction . . . 39
4.3.1 Basic Syntax Differences . . . 40
4.3.2 Data Types . . . 40
4.3.3 Case . . . 42
4.3.4 REXX Instructions . . . 42
4.3.5 Function Calls . . . 45
4.3.6 Subroutines and User Defined Functions . . . 47
4.3.7 Exit or Return . . . 49
4.3.8 Stems - Array Variables - Indexed Strings . . . 50
4.3.9 The main() Method - Input Parameters . . . 52
4.3.10 Comparing NetRexx to Object Oriented REXX . . . 53

Chapter 5. AboutFrame, a Reusable Class . . . 55
5.1 The AboutFrame Picture . . . 55
5.2 What is AboutFrame? . . . 55
5.3 AboutFrame: User Interface . . . 56
5.4 AboutFrame: Program Interface . . . 56
5.4.1 Approach with Classic Languages . . . 56
5.4.2 An OO Solution . . . 56
5.5 Classes and Methods . . . 57
5.5.1 Class - What is it? . . . 57
5.5.2 Methods - What are they? . . . 58
5.5.3 Variables in the Class . . . 60
5.6 AboutFrame: the Class Definition . . . 60
5.6.1 AboutFrame: Overview of the Program . . . 61
5.6.2 AboutFrame Section One: The Class Itself . . . 63
5.6.3 AboutFrame Section Two: The Constructor Method . . . 64
5.6.4 AboutFrame Section Three: Other Methods . . . 66
5.6.5 AboutFrame Section Four: Event Classes . . . 67
5.6.6 Avoiding Empty Frames . . . 73

Chapter 6. Reading and Writing Files from NetRexx . . . 75
6.1 Reading BFS Character Data Files . . . 75
6.1.1 Reading CMS Character Data Files . . . 75
6.2 Reading from the console . . . 77
6.2.1 Useful Control Sequences . . . 77
6.3 Writing BFS Character Data Files . . . 78
6.3.1 Writing CMS Character Data Files . . . 78
6.4 Working With Binary Files . . . 79

Chapter 7. Code Pages - ASCII <> EBCDIC Issues . . . 81
7.1 History, Experience . . . 81
7.2 Background Information - Codepages . . . 82
7.3 Internationalization . . . 83
7.3.1 Streams? . . . 83
7.3.2 Java IO Support . . . 83
7.4 VM Java Codepage . . . 84
7.4.1 Solution for Client Server Programs . . . 84

7.5 IBM Network Station and Codepages . . . 85

Chapter 8. TCP/IP Networking . . . 87
8.1 Translating between EBCDIC and ASCII . . . 87
8.1.1 readLine() and printLn() . . . 87
8.2 Simple TCP/IP Client . . . 88
8.3 Simple TCP/IP Server . . . 88
8.3.1 Extending the Server . . . 91
8.3.2 Starting the Server . . . 91

Chapter 9. Java and CMS . . . 93
9.1 Executing non-Java Programs . . . 93
9.1.1 Using Runtime.exec() . . . 93
9.1.2 Using JNI . . . 94
9.2 The cms.util Package . . . 94
9.3 Running CMS Execs . . . 95
9.3.1 The CMSRexx Class . . . 95
9.4 Running CMS Pipelines with NetRexx . . . 96
9.4.1 fitting *>java . . . 96
9.4.2 fitting *<java . . . 97
9.4.3 The CMSPipe Class . . . 97
9.5 Installation Instructions . . . 100

Chapter 10. The GUIMON Sample Program . . . 101
10.1 GUIMON - Pictures . . . 101
10.2 GUIMON - Installation Instructions . . . 103
10.2.1 Installing the GUIMON Monitor . . . 103
10.2.2 Installing the GUIMON Client . . . 105
10.2.3 Installing the GUIMON Server . . . 106
10.2.4 Installing Client and Server Files . . . 107
10.3 GUIMON - Functional Overview . . . 110
10.3.1 GUIMON - the Monitor . . . 110
10.3.2 GUIMON - the Server . . . 111
10.3.3 GUIMON - the Client . . . 112
10.4 GUIMON - the Client-Server Communication . . . 113
10.4.1 Request Formats . . . 113
10.5 GUIMON Record Format Requirements . . . 117

Chapter 11. Running NetRexx and Java Applications on a Network Station 121
11.1 Network Computing - Extending VM/ESA Resources into the Network 121
11.2 VM/ESA as a Network Station Server . . . 122
11.3 Support Delivery Mechanism . . . 122
11.4 Hardware Requirements for VM/ESA . . . 122
11.5 Software Requirements for VM/ESA . . . 122
11.6 Major Steps to Install VM/ESA Network Station Code . . . 123
11.6.1 Download the Network Station Code . . . 123
11.6.2 Prepare for the Installation of the Network Station Client Code . . 123
11.6.3 Plan the Byte File System File Space Structure . . . 125
11.6.4 IBM Network Station Browser for VM/ESA . . . 126
11.6.5 IBM Network Station Customization . . . 127
11.7 Java Programs on the IBM Network Station . . . 127
11.8 Setting up to Run Java and NetRexx Programs . . . 128
11.8.1 How to Copy the NetRexx Runtime Environment . . . 128
11.9 Starting a Java or NetRexx Program on your IBM Network Station . . 130
11.10 How to Tailor the Local File System . . . 131

11.10.1 Performance Considerations 133
11.11 Using NSM to Add a Java Application to the Menu Bar 133
11.12 Starting the GuiMon Application on the Network Station 136
11.12.1 Login to the Network Station 137
11.12.2 Start GuiMon from the Menu Bar 137
11.12.3 Summary 139

Appendix A. Frequently Asked Questions 141
A.1 NullPointerException - General Problem 141
A.2 NullPointerException - With Compound Variables 141
A.3 NetRexx: No Data Type Problems Anymore? 142
A.4 Error Messages Not Always Very Accurate 142
A.5 File Not Found 142
A.6 Threads Class Not Found 142
A.7 External Link Files Not Found 143
A.8 Reading Java Abend Messages 143
A.9 Virtual Storage Requirements 143
A.10 Killing the Java Virtual Machine in VM 144
A.11 Runtime Problems 145
A.12 Installation Problems 145

Appendix B. NetRexx Language Quick Start 147
B.1 Introduction 147
B.2 NetRexx Programs 147
B.3 Expressions and Variables 148
B.4 Control Statements 149
B.5 NetRexx Arithmetic 150
B.6 Doing Things with Strings 151
B.7 Parsing Strings 151
B.7.1 Parsing into Words 152
B.7.2 Literal Patterns 152
B.7.3 Positional Patterns 152
B.8 Indexed Variables 152
B.9 Arrays 153
B.10 Tracing 154
B.11 Exception and Error Handling 154
B.12 Things that aren′t Strings 155
B.12.1 Programs are Classes 156
B.13 Extending Classes 156
B.13.1 Optional Arguments 158
B.14 Binary Types and Conversions 158
B.14.1 Binary Types in Practice 159
B.15 Summary and Information Sources 159

Appendix C. Special Notices 161

Appendix D. Related Publications 163
D.1 International Technical Support Organization Publications 163
D.2 Redbooks on CD-ROMs 163
D.3 Other IBM Publications 163
D.4 At Your Local Bookstore 164
D.5 On the Web 164

How to Get ITSO Redbooks 165
How IBM Employees Can Get ITSO Redbooks 165

How Customers Can Get ITSO Redbooks 166
IBM Redbook Order Form 167

Glossary 169

List of Abbreviations 183

Index 185

ITSO Redbook Evaluation 187

Figures

1. GUIMON - The Sample NetRexx VM/ESA Application 1
2. SFS file pool disk structure 4
3. VMSYSU File Pool Server Directory 5
4. VMSYSR File Pool Server Directory 6
5. VMSYS File Pool Server Directory 6
6. Special Characters 9
7. Typical BFS ROOT Tree Structure 10
8. Java Architecture 12
9. AboutFrame, Displays Information 55
10. AboutFrame, Program Overview 61
11. AboutFrame, Class Definition and Properties 63
12. AboutFrame, Constructor Method 64
13. AboutFrame, Other Methods 66
14. AboutFrame, Event Classes 67
15. AboutFrame, Event Classes 68
16. AboutFrame, Event Handling with More Classes 70
17. AboutFrame, a Main Method 72
18. AboutFrame, Empty Frame Problem 73
19. readAfile.nrx 75
20. writeAfile.nrx 78
21. factTable.nrx 79
22. Socket Client Program 81
23. Socket Server Program 82
24. Simple NetRexx TCP/IP Client 88
25. TCP/IP Server in NetRexx 89
26. ServerHandler.nrx, Extending the TCPServer.nrx Class 91
27. GUIMON Prompting for Host Address 101
28. GUIMON Listing the Performance Information Found 102
29. GUIMON Plotting a Performance Variable 102
30. Extract of GUIMON's Server Code 115
31. Extract of GUIMON's Server Code 116
32. Extract of GUIMON's Server Code - INDICATE Request 117
33. Extract of GUIMON's Server Code - PERFLOG DESCRIBE Example . . 119
34. Byte File System Structure 125
35. Copy the NetRexx Runtime Environment Using the Shell 129
36. Copy the Codepage 1047 Class Files Using the Shell 129
37. NETCPY EXEC to Copy the NetRexx and Cp1047 Classes 130
38. Network Configuration 131
39. IBM Network Station Manager Login Screen 134
40. IBM Network Station Manager Administrator Main Menu 134
41. IBM Network Station Manager Startup Menus 135
42. IBM Network Station Manager Java Application Menu Items 136
43. IBM Network Station Login Screen 137
44. Built by GuiMon Classes Loaded from Germany 138
45. Built by the AboutFrame classes Loaded from Sweden 138

Tables

1. Some SFS and BFS Tasks 11
2. Useful control sequences 77

© Copyright IBM Corp. 1998

Preface

Java is the hot new programming language for the nineties and beyond and for developing a whole new category of Internet aware applications. NetRexx ′improves′ upon Java by blending the best features of classic REXX with the object model and semantics of Java and the Java Virtual Machine execution environment. Together, these languages provide powerful new tools for enabling VM/ESA systems to fully participate on both the Internet and on organizations′ internal ′intra-nets′. Coupled with the new IBM Network Station, VM/ESA can now provide a powerful solution for bringing network aware applications to large numbers of simultaneous users in a very cost effective manner.

This redbook describes how to install and configure both Java and NetRexx using the IBM provided installation files and scripts. It also covers a selection of VM tools that make using NetRexx and Java easier and less trouble-prone to the applications developer. There is a detailed comparison of NetRexx to classic REXX to help guide VM/REXX programmers through the similarities and differences and introduce them to the novel capabilities of NetRexx. Also included are hints and tips on good NetRexx design and a wealth of examples of NetRexx application programming techniques in the CMS environment.

The book culminates with a complete demonstration application, comprised of a multithreaded CMS server written entirely in NetRexx and a graphical client, written in a mixture of NetRexx and Java, that may be run on IBM Network Stations or other Java platforms, which provides various graphical views of VM performance statistics. Detailed instructions for configuring a network station environment to support the execution of such VM/ESA-based applications are also included.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Böblingen Center.

The authors of this redbook are:

Kris Buelens is a Systems Engineer in IBM Belgium, professional services. All his experience, since 1978, is based on field work, initially VM and VSE, the last 15 years VM only.

Bengt Heijnesson is an IT specialist in IBM Sweden professional services. He has 12 years of experience in the VM arena. His areas of expertise are CP, CMS (application programming in assembler and REXX), TCP/IP and Network Stations.

Dave Jones is a software developer for Velocity Software, Inc., based in Houston, Texas. His areas of expertise include application development using PL/I, REXX, and CMS Pipelines as well as VM/ESA performance monitoring and analysis.

Salvador Torres is an IT Specialist in IBM USA, Global Services. He has 15 years of experience in the VM arena. He holds a degree in Computer Science from Loyola University in Chicago. His areas of expertise include CMS, OV/VM, and networking products. He has programmed extensively in REXX, Pipelines and C.

© Copyright IBM Corp. 1998

The residency that produced this redbook was coordinated by:

Stephen Record
International Technical Support Organization, Böblingen Center

Thanks to the following people for their invaluable contributions to this project:

Marci Beach
IBM Endicott

Mike Cowlishaw
IBM Hursley

Jack Crast
IBM Endicott

Mike Donovan
IBM Endicott

Tim Preece
IBM Hursley

Günter Schmid
IBM Böblingen

Reinhold Walter
IBM Böblingen

Gudrun Wiedemann
International Technical Support Organization, Böblingen Center

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in "ITSO Redbook Evaluation" on page 187 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

For Internet users	http://www.redbooks.ibm.com/
For IBM Intranet users	http://w3.itso.ibm.com/

- Send us a note at the following address:

  redbook@us.ibm.com

Chapter 1. Introduction

This redbook introduces the new programming environments NetRexx and Java for VM/ESA Version 2, Release 3. The book focuses on these areas:

- deploying NetRexx and Java on VM/ESA
- approaching NetRexx from a REXX background
- developing VM/ESA-based applications in Java and NetRexx

First, we provide an overview of NetRexx, Java, and their key components. The NetRexx and Java implementations under VM/ESA V2.3 will be covered as well, along with details of VM-specific considerations. We also describe how to install and configure NetRexx and Java on VM/ESA. Included in this section is information on how to set up and configure the SFS and BFS components of VM/ESA, which NetRexx and Java require. We further discuss a number of helpful tools and techniques developed during the residency for managing the Java and NetRexx program development and execution environment under CMS.

With that foundation established, we then provide a guide for classic REXX programmers to the NetRexx language and some of the most useful principles and techniques of object oriented programming. After that we explore various important NetRexx programming techniques for the CMS environment, and provide numerous illustrative examples.

Finally, we present a complete VM/ESA application built using NetRexx and Java. Both the client side graphical user interface and the server side CMS application are coded entirely in NetRexx and Java The application presents to the user a chart of a VM/ESA system′s performance data and allows the user to “point and click” to select different data sets to view. Communication between the server and the client is accomplished using the native Java interfaces to TCP/IP sockets. This sample application, known as GUIMON, is depicted in the following diagram:

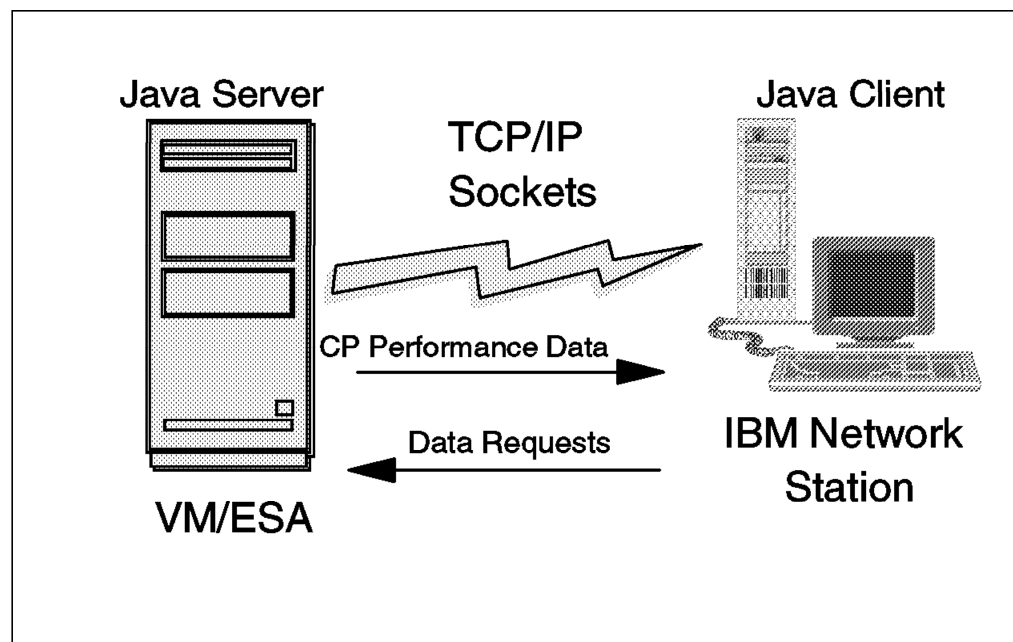


Figure 1. GUIMON - The Sample NetRexx VM/ESA Application

© Copyright IBM Corp. 1998

Throughout the residency we used IBM Network Stations as our end-user workstations, both as emulated 3270 terminals for logging on to VM/ESA and as the client platform for the GUIMON graphical user interface. This redbook also includes a detailed discussion of how to configure both the IBM Network Station and the VM/ESA server system in order to make effective use of the IBM Network Station as a NetRexx program development and execution platform.

All of the programs developed during the residency and described in this redbook are available for download on the VM/ESA download page on the World Wide Web at URL `http://www.vm.ibm.com/download/`. See 2.7.1.5, "Download and Install the SG245148 Package" on page 19 for information on obtaining the entire package of sample programs for your VM/ESA system.

Chapter 2. Overview of NetRexx and Java on VM/ESA

2.1 Java, NetRexx, OpenEdition, and the BFS

Java has its roots in systems that support a Unix-style hierarchical tree-structured file directory. Therefore, it's not surprising to find that Java (and by extension, NetRexx) requires access to a hierarchical file system in order to function correctly. Such a file store is provided by the Byte File System (BFS), an integral part of OpenEdition VM/ESA. BFS files and directories are stored in CMS file pools that are managed by the Shared File System (SFS). This chapter explains what you need to know about the BFS in order to be able to deploy and use Java and NetRexx effectively on VM/ESA.

2.2 Overview of the SFS

The Shared File System (SFS) is the component of VM/ESA that provides a hierarchical, managed file system to CMS users, as opposed to the traditional CMS minidisk file system. Some of the advantages to using the SFS instead of the traditional CMS minidisk file system are:

- multiple ***safe*** READ/WRITE sharing of files;
- better data organization in a hierarchical tree structure;
- better utilization of existing DASD space, due to the elimination of wasted space on minidisk volumes;
- cross-system file access, similar to that provided by NFS;
- and, of course, the ability to support the BFS.

Information on the following SFS topics is important in installing and deploying the IBM Network Station, Java, and NetRexx effectively. This information can be found in *VM/ESA CMS File Pool Planning, Administration, and Operation*, SC24-5751; an overview is given here for your convenience.

2.2.1 SFS Servers and File Pools

2.2.1.1 Servers

An SFS server is an ordinary service virtual machine, usually autologged by the system at IPL time, that executes the SFS “data base engine” to manage CMS files using a set of *control*, *log*, and *user data* disks as shown in Figure 2 on page 4.

© Copyright IBM Corp. 1998

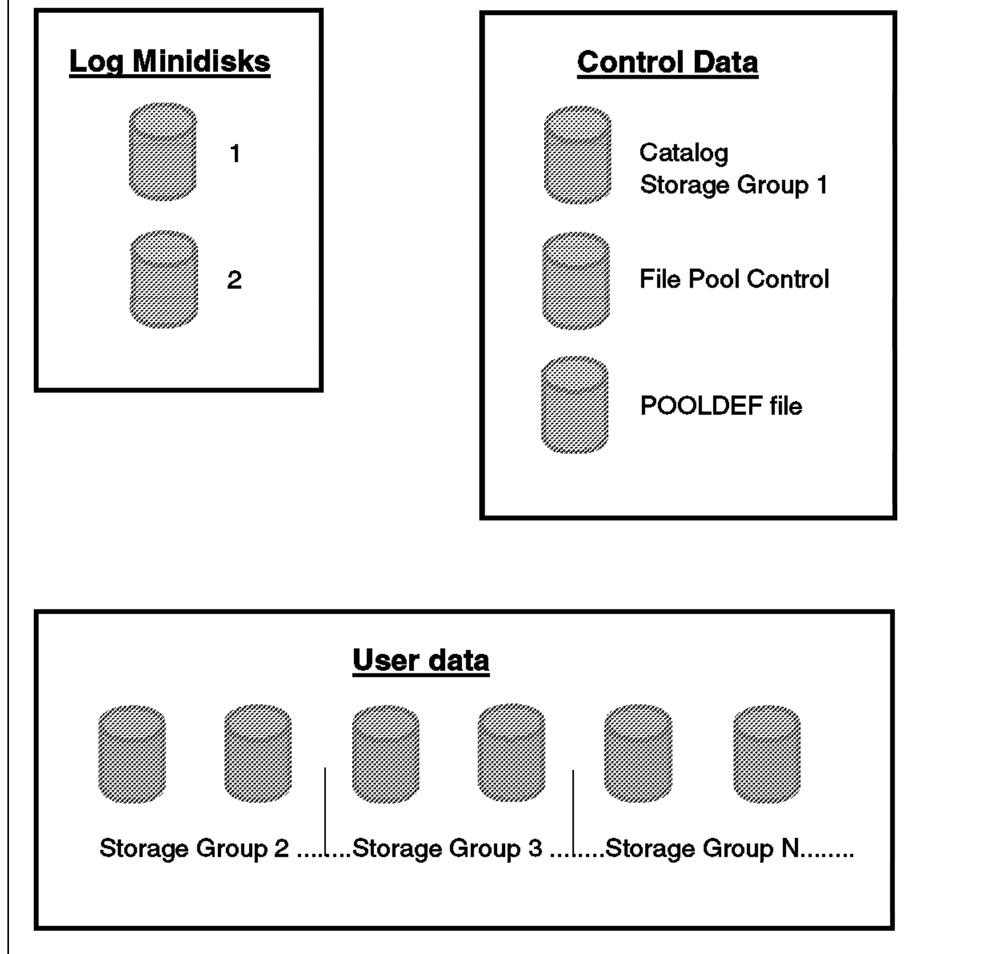


Figure 2. SFS file pool disk structure

A file pool is a collection of minidisks that are managed by the SFS server virtual machine. The file pool contains:

Storage Groups 2, 3, n: minidisks within the file pool that contain the users′ data files.

Catalog Storage Group: the storage group within the file pool that contains the catalog and other information on the users′ authorizations and user directories, similar to the File Status Table (FST) entries for a regular CMS minidisk.

Log Minidisks: A set of independent minidisks holding log information; maintained in a “fail-safe” dual-write method allowing changes to user files to be either committed or rolled back.

Control Data: Minidisk(s) containing information about the DASD space allocations and availability of data blocks in the file pool.

It is ***critically*** important that the appropriate file pool parameters and minidisk number and sizes be correctly defined when the SFS file pool is built. Detailed installation planning instructions and suggested configuration parameter values

may be found in Chapter 20, “File Pool Server Start-up Parameters,” of the *VM/ESA CMS File Pool Planning, Administration, and Operation*, SC24-5751, manual referenced above.

An SFS server can also make file pools available to remote VM/ESA systems through the services of the Inter-System Facility for Communications (ISFC), the Transparent Service Access Facility (TSAF), and APPC/VM VTAM Support (AVS). For complete information on VM connectivity capabilities, refer to *VM/ESA Connectivity Planning, Administration, and Operation*, SC24-5756.

2.2.1.2 Standard File Pool Names

VM/ESA Version 2 Release 3 is shipped with a number of SFS file pools ready to be installed by the installation process. Three of these predefined file pools are:

VMSYS SFS file pool for system resources; for example, program products

VMSYSU SFS file pool for general user data files and directories

VMSYSR CRR recovery server file pool for coordinated resource recovery across multiple file pools and improved SFS performance

Since these file pools all have names that start with VMSYS... they are **not** eligible for access by remote VM systems. Here is the USER DIRECT entry for the VMSYSU file pool server, VMSERVU, that we use here at the ITSO Center in Böblingen for this residency.

```
USER VMSERVU VMSERVU 32M 32M BG
 ACCOUNT 1 VMSERVU
 MACH XC
 OPTION MAXCONN 2000 NOMDCFS APPLMON ACCT QUICKDSP SVMSTAT
 SHARE REL 1500
 XCONFIG ADDRSPACE MAXNUMBER 100 TOTSIZE 8192G SHARE
 XCONFIG ACCESSLIST ALSIZE 1022
 IUCV ALLOW
 IUCV *IDENT RESANY GLOBAL
  IPL CMS
 POSIXOPT SETIDS ALLOW
 CONSOLE 009 3215 T MAINT
 SPOOL 00C 2540 READER *
 SPOOL 00D 2540 PUNCH A
 SPOOL 00E 1403 A
 LINK MAINT 190 190 RR
 LINK MAINT 193 193 RR
 LINK MAINT 19D 19D RR
 LINK MAINT 19E 19E RR
 MDISK 191 3380 0645 003 230W01 MR RSERVER  WSERVER
 MDISK 301 3380 0995 010 230W01 WR RCONTROL WCONTROL
 MINIOPT NOMDC
 MDISK 302 3380 1005 016 230W01 WR RLOG1    WLOG1
 MINIOPT NOMDC
 MDISK 303 3380 1021 016 230W01 WR RLOG2    WLOG2
 MINIOPT NOMDC
 MDISK 304 3380 1037 003 230W01 WR RCATALOG WCATALOG
 MDISK 305 3380 1040 007 230W01 WR RDATA    WDATA
 MDISK 306 3380 0182 138 230W01 WR RDATA    WDATA
 MDISK 307 3380 0053 008 230W01 WR RCATALOG WCATALOG
 MDISK 308 3380 0622 0300 230W02 WR RDATA    WDATA
 MDISK 309 3380 0923 0020 230W02 WR RCATALOG WCATALOG
```

Figure 3. VMSYSU File Pool Server Directory

for the VMSYSR file pool server, VMSERVR,

```
USER VMSERVR VMSERVR 32M 32M BG
 ACCOUNT 1 VMSERVR
 MACH XA
 OPTION MAXCONN 2000 APPLMON ACCT QUICKDSP SVMSTAT
 SHARE REL 1500
 IUCV ALLOW
 IUCV *IDENT RESANY GLOBAL
  IPL CMS
 CONSOLE 009 3215 T MAINT
 SPOOL 00C 2540 READER *
 SPOOL 00D 2540 PUNCH A
 SPOOL 00E 1403 A
 LINK MAINT 190 190 RR
 LINK MAINT 193 193 RR
 LINK MAINT 19D 19D RR
 LINK MAINT 19E 19E RR
 MDISK 191 3380 0648 002 230W01 MR RSERVER  WSERVER
 MDISK 301 3380 1047 002 230W01 WR RCONTROL WCONTROL
 MINIOPT NOMDC
 MDISK 302 3380 1049 001 230W01 WR RLOG1    WLOG1
 MINIOPT NOMDC
 MDISK 303 3380 1050 001 230W01 WR RLOG2    WLOG2
 MINIOPT NOMDC
 MDISK 304 3380 1051 002 230W01 WR RCATALOG WCATALOG
 MINIOPT NOMDC
 MDISK 305 3380 1053 001 230W01 WR RDATA    WDATA
 MINIOPT NOMDC
 MDISK 306 3380 1054 002 230W01 WR RCRRLOG1 WCRRLOG1
 MINIOPT NOMDC
 MDISK 307 3380 1056 002 230W01 WR RCRRLOG2 WCRRLOG2
 MINIOPT NOMDC
```

Figure 4. VMSYSR File Pool Server Directory

and, lastly, for the VMSYS file pool server, VMSERVS:

```
USER VMSERVS VMSERVS 32M 32M BG
 ACCOUNT 1 VMSERVS
 MACH XC
 OPTION MAXCONN 2000 NOMDCFS APPLMON ACCT QUICKDSP SVMSTAT
 SHARE REL 1500
 XCONFIG ADDRSPACE MAXNUMBER 100 TOTSIZE 8192G SHARE
 XCONFIG ACCESSLIST ALSIZE 1022
 IUCV ALLOW
 IUCV *IDENT RESANY GLOBAL
  IPL CMS
 POSIXOPT SETIDS ALLOW
 CONSOLE 009 3215 T MAINT
 SPOOL 00C 2540 READER *
 SPOOL 00D 2540 PUNCH A
 SPOOL 00E 1403 A
 LINK MAINT 190 190 RR
 LINK MAINT 193 193 RR
 LINK MAINT 19D 19D RR
 LINK MAINT 19E 19E RR
 MDISK 191 3380 0642 003 230W01 MR RSERVER  WSERVER
 MDISK 301 3380 0650 005 230W01 WR RCONTROL WCONTROL
 MINIOPT NOMDC
 MDISK 302 3380 0655 005 230W01 WR RLOG1    WLOG1
 MINIOPT NOMDC
 MDISK 303 3380 0660 005 230W01 WR RLOG2    WLOG2
 MINIOPT NOMDC
 MDISK 304 3380 0665 030 230W01 WR RCATALOG WCATALOG
 MDISK 305 3380 0695 300 230W01 WR RDATA    WDATA
 MDISK 306 3380 2370 001 230RES WR RDATA    WDATA
 MDISK 307 3380 2536 066 230RES WR RDATA    WDATA
 MDISK 308 3380 0423 074 230W01 WR RDATA    WDATA
 MDISK 309 3380 2371 111 230RES WR RDATA    WDATA
 MDISK 30A 3380 2107 162 230RES WR RDATA    WDATA
```

Figure 5. VMSYS File Pool Server Directory

The amount of DASD space allocated to these VM users′ minidisks is more than sufficient to support all of the Java, NetRexx, and IBM Network Station development and testing this Redbook documents. They′re good starting values for your site, too.

In each of these USER DIRECT directory entries, the 191 minidisk holds the POOLDEF file, the 301 minidisk is the file pool control minidisk, minidisks 302 and 303 hold the log files, and minidisk 304 is the file pool catalog (Storage Group 1) minidisk. In VMSERVU, because the number of user data files and directory can be quite large in a file pool, the 307 and 309 minidisks are also allocated to the file pool catalog storage group as well. All of the other minidisks defined in the directory entries are allocated to hold the user data files and directories in Storage Groups 2 through N.

2.2.1.3 SFS Backup and Restore Issues

Due to the amount and value of data stored in a typical site′s SFS/BFS file pool, some careful thought should be given to file pool backup and restore issues. We recommend that both the user data and control data components of the SFS be backed up at regular intervals. You can use either the native backup/restore facilities that SFS provides or you can use some other non-file pool server aware backup technique, such as the standard VM/ESA facility DDR (DASD Dump Restore) program.

Obviously, you can buy a product to manage SFS backups; such solutions often offer incremental backup facilities. Here we only mention the SFS backup methods natively available in any VM/ESA system.

If you elect to back up only the user data component, (storage groups 2, 3,...N), and not the control data component (catalog, file pool control disks, and POOLDEF disk), and you lose either the control disk or the storage group 1 (catalog) disks, you will be forced to regenerate your file pool and restore **every** user storage group. For large SFS collections, this can be a very lengthy process. On the other hand, if you only back up the control component and not the user data component, you will lose the entire file pool if any one disk in the SFS becomes corrupted. There is no way to recover from this situation.

The recommended approach to backing up and restoring an SFS file pool is to use the native SFS BACKUP and RESTORE facilities. SFS provides the FILEPOOL BACKUP and FILEPOOL RESTORE administrator commands to back up and restore SFS file pools. Use the FILEPOOL BACKUP command to make backups of the user data areas (storage groups 2 through N), and use the FILEPOOL CONTROL BACKUP command to make backups of the control disks, catalog storage group 1, and the POOLDEF disk. One important feature of using these commands is that the file pool can still be made available to end users (although in read-only mode) while the backup is taking place. Another feature is that individual users or files can be restored (using the FILEPOOL RELOAD command). These commands and their operands are documented in the manual *CMS File Pool Planning, Administration, and Operation*, SC24-5751.

If you elect to use a non-file pool server aware backup method, such as DDR, you should be aware of the following issues:

- the SFS file pool must be completely shutdown while the backup is made; the end user cannot have even read-only access to the data

- you must back up **all** of the file pool minidisks on which the file pool resides **at the same time**. You cannot back up half of the minidisks today, start the file pool server for normal operations, and then back up the remaining minidisks tomorrow.

A file pool is a single logical entity. The data on the file pool minidisks are logically intertwined in a complex manner. Without using the native SFS file pool backup and restore commands (or a similar product) that are aware of the logical interrelationships of the data, the file pool must be backed up or restored as a unit. It is not possible to restore only a portion of the file pool.

2.3 Overview of the BFS

The redbook *OpenEdition for VM/ESA Implementation and Administration Guide*, SG24-4747, contains an outstanding description of the whole OpenEdition environment. For your convenience a shorter overview is included here.

OpenEdition for VM/ESA provides a POSIX compliant file system called the Byte File System (BFS). Unlike the conventional record-oriented CMS file systems, both minidisk-based and SFS, the BFS treats files as nothing more than an ordered collection of bytes. It imposes no other structure on the data stored in files. The POSIX concept of a file system was derived conceptually from the file systems first developed for Unix systems. The BFS has different semantics, file naming conventions, and file structures from the conventional CMS file systems.

The BFS allows files to be created and used in a Unix-style format. Like files in the SFS, BFS files are stored in CMS file pools. A BFS file space can be in the same file pool as SFS file spaces, and more than one BFS file space can be enrolled in the same file pool.

The POSIX defined interfaces for the "C" programming language are provided as callable library routines in the VM/ESA provided Language Environment runtime library, SCEERUN LOADLIB. For other programming environments and languages (for example, REXX, PL/I, Assembler), access to POSIX functions is provided by a set of Callable Services Library (CSL) routines. OpenEdition for VM/ESA also provides a set of CMS Pipeline stages and a set of OPENVM subcommands to support the BFS. Some native CMS commands (for example, XEDIT) can also support access to BFS files and directories, using extensions to the CMS record file system interface.

2.3.1 POSIX Terminology

Here are the definitions of some of the more common POSIX terms that arise when using the BFS.

Path name
A character string that identifies a path to a file or directory. OpenEdition supports path names up to 1023 characters long.

Relative path name
A path name that identifies the path to a file or directory from the current working directory. Relative path names do not begin with a slash (/).

Absolute path name
A path name that identifies the path to a file or directory from the file system root. Absolute path names always begin with a slash (/).

File name A character string that names a file within a directory. OpenEdition supports file names up to 255 characters long.

Directory A special file that contains directory entries. Directory entries must be unique within a given directory, but different directory entries can associate different names to the same file.

Current (or working) directory The directory associated with a process (for example, the shell) that is used to resolve relative path names.

The characters used in file and path names should be drawn from the POSIX portable character set, but this is not enforced by OpenEdition for VM/ESA. The portable character set consists of:

- the upper case letters A-Z.
- the lower case letters a-z.
- the digits 0-9.
- the special characters dot (.), underscore (_) and hyphen (-).

A compliant name cannot start with a hyphen. A valid POSIX file name might be:

```
karen.nrx
```

and its corresponding absolute path name might look like this:

```
/home/dave/netrexx/examples/karen.nrx
```

Note that everything up to and including the last slash (/) in the above example is part of the directory path, and everything after the last slash is the file name itself. When the current directory is set to `/home/dave` the relative path name is:

```
netrexx/examples/karen.nrx
```

Because the slash (/) is the path separator character, it cannot be used in a file name. Also, because the standard OpenEdition shell interpreter assigns special meanings to the following characters, it's not a good idea to use them in file or directory names either:

```
(blank) * # / \ < > | & $ ? ( ) { }
```

Figure 6. Special Characters

Make files names easy to remember. Unlike VM/ESA, which restricts both file names and file types to eight (or less) characters, OpenEdition permits file names to be up to 255 characters long. Use the dot (.) or the underscore (_) to make file names more readable and easy to remember; for example:

```
will_jones_birthday_gift_list
```

Over the years, users of Unix-style operating systems have developed a set of conventions for arranging the initial directories off the main or **root** directory. The POSIX BFS follows these conventions as implemented by OpenEdition for VM/ESA, so you are likely to see a first-level directory structure that looks something like this:

```
/           (← the root directory)
  bin       (contains executable commands)
  dev       (support for hardware devices)
  etc       (contains administration files, the toolbox)
  home      (contains user directories and files)
  lib       (symbolic link to /usr/lib libraries and shared libraries)
  opt       (contains DCE administration files)
  tmp       (contains system temporary files and work areas)
  u         (symbolic link to /home)
  usr       (contains system executable files, administration files, etc.)
  var       (contains log files, security and spool files, etc.)
```

Figure 7. Typical BFS ROOT Tree Structure

2.3.2 Directory Entries for POSIX BFS Usage

In order for a VM user ID to be able to use the BFS and the OpenEdition for VM/ESA Shell and Utilities, some POSIX directory control statements may need to be added in the user directory entry. These directory control statements are:

- POSIXINFO
- POSIXOPT
- POSIXGLIST

2.3.2.1 The POSIXINFO Statement

The POSIXINFO statement is probably the most important of these three control statements. It defines the user's POSIX user database information, such as the POSIX user ID associated with this VM user ID, the POSIX group ID, the initial working directory, the initial user program (that is, POSIX shell) to run, and the file system mount point. For example, the VM user ID SRRES3 directory entry has the following POSIXINFO statement in it:

```
POSIXINFO UID 0 GNAME staff IWDIR /home/dave/
```

This sets user SRRES3's POSIX user ID (*uid*) to 0 (zero), his POSIX group name (*gname*) to *staff*, and the initial working BFS directory (*iwdir*) to */home/dave/*. See the manual *VM/ESA Planning and Administration*, SC24-5750, for more details on the operands and syntax of the POSIXINFO statement.

2.3.2.2 The POSIXOPT Statement

The POSIXOPT statement specifies option settings that are related to a VM user ID's POSIX capabilities. The capabilities include authorization to query and set certain POSIX process and database information. For example, to allow a VM user ID to set any other users' POSIX security values and query any other users' POSIX database information, the following POSIXOPT statement would be placed in the user's directory entry:

```
POSIXOPT SETIDS ALLOW QUERYDB ALLOW
```

See the manual *VM/ESA Planning and Administration*, SC24-5750, for more details on the operands and syntax of the POSIXOPT statement.

2.3.2.3 The POSIXGLIST Statement

The POSIXGLIST statement specifies which POSIX groups a user ID is a member of. Each group can be specified either by the *group id (gid)* or by the *group name (gname)*. To specify that a user ID is a member of gid 0 and gnames *staff* and *operations*, code the following POSIXGLIST statement in the user's directory entry:

```
POSIXGLIST GID 0 GNAMES staff operations
```

See the manual *VM/ESA Planning and Administration*, SC24-5750, for more details on the operands and syntax of the POSIXGLIST statement.

2.4 Some Common SFS and BFS Commands

Some readers may be new to BFS and even SFS. Therefore, the commands listed in Table 1 may be useful.

Table 1. Some SFS and BFS Tasks

Task	**SFS command**	**BFS command**
Enroll a user	ENROLL USER xyz	ENROLL USER xyz (BFS
Create a directory	CREATE DIR dirid	OPENVM CREATE DIR path
Erase a directory	ERASE dirid	OPENVM ERASE path
Create an alias	CREATE ALIAS ...	OPENVM CREATE LINK ... OPENVM CREATE EXTLINK ... OPENVM CREATE SYMLINK ...
Define default directory	ACCESS dir A	OPENVM SET DIR path
Copy a file	COPYFILE ...	OPENVM GETBFS ... OPENVM PUTBFS ...
Erase a file	ERASE fn ft fm	OPENVM ERASE path
Move a file or directory	RELOCATE ...	OPENVM RENAME ...
Xedit a file	XEDIT fn ft fm	XEDIT path (NAMET BFS
Set permissions	GRANT AUTH ... REVOKE AUTH ...	OPENVM PERMIT ...
Get access to files	ACCESS dir fm	OPENVM MOUNT ...
Drop access to files	RELEASE fm	OPENVM UNMOUNT ...

To get more information about SFS commands, issue **HELP CMS cmd** or **HELP SFSADMIN cmd**.

To get more information about BFS commands, issue **HELP OPENVM cmd** or **HELP OPENVM MENU**.

2.5 The Java Environment Under VM

As shown in Figure 8, the Java architecture is one that

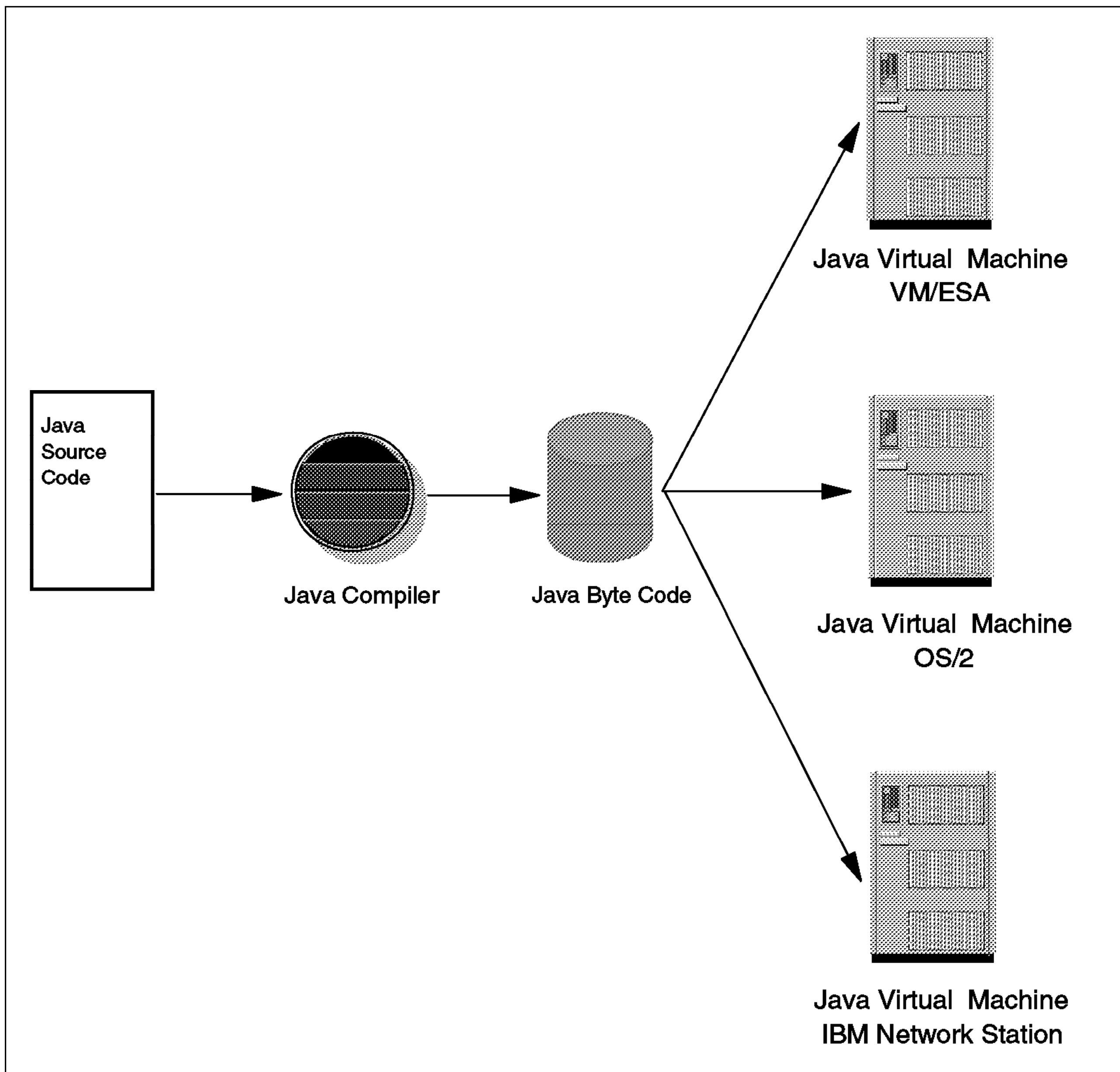


Figure 8. Java Architecture

takes as input a program coded in Java (or, in our case, one coded in NetRexx as well) and produces a machine and operating system independent output termed *Java byte code*. This byte code, in turn, is interpreted by a *Java virtual machine* program to produce the desired output. Since there are now many implementations of the Java virtual machine for many different machines and operating systems, the byte code has come to be called *machine independent* or "write once, run anywhere" code.

The Java virtual machine that interprets the Java byte code is nothing more sophisticated than a large "C" program. The VM version of the Java virtual machine is written to conform to POSIX standards. It assumes that the

underlying hardware and software can provide the services (for example, file I/O, memory allocation, and process management) defined by the POSIX standard. In VM/ESA, the POSIX support is provided by the OpenEdition feature and by the BFS; thus, in VM the Java and NetRexx packages are installed into the BFS, using the facilities of OpenEdition.

2.6 SFS and BFS Directory Structures

Even though both SFS and BFS file spaces are managed by an SFS server, there are quite some organizational and usage differences.

2.6.1 An SFS File Space

With an SFS, when an end-user is enrolled, he gets what is called a "file space." The following command enrolls user KRIS in the SFS named VMSYS, and allows him to use up to 10000 4K data blocks.

```
ENROLL USER KRIS VMSYS (BLOCKS 10000
```

The end-users organize their file space as they want. From the directory names, it is clear who owns what. The DIRLIST command can be used to see the directories of a file space. For example DIRLIST VMSYS:KRIS. could produce:

```
 KRIS   DIRLIST  A0  V 319  Trunc=319 Size=5 Line=1 Col=1 Alt=0
Cmd   Fm Directory Name
      A  VMSYS:KRIS.
      -  VMSYS:KRIS.BFSLIST_AND_CO
      Z  VMSYS:KRIS.GUIMON
      -  VMSYS:KRIS.GUIMON.NEW
      -  VMSYS:KRIS.NRTOOLS


1= Help      2= Refresh  3= Quit   4= Sort(fm)   5= Sort(dir)  6= Auth
7= Backward  8= Forward  9=       10=           11= Filelist  12= Cursor

====>
```

Mostly files in an SFS directory are used after having accessed it: with the ACCESS command a letter is assigned to the directory. This letter is used to refer to the files in the directory and also defines the place in the CMS search order. The QUERY ACCESSED command can be used to list the search order

```
 access vmsys:srecord.jnr_res.samples D (forcerw
Ready;
 q accessed
Mode  Stat     Files  Vdev  Label/Directory
A      R/W       175  DIR   VMSYS:KRIS.
B      R/W       233  DIR   VMSYS:SRECORD.JNR_RES
C      R/O       195  DIR   VMSYS:SRECORD.REDBOOK
D      R/W        93  DIR   VMSYS:SRECORD.JNR_RES.SAMPLES
E      R/O       198  100   ICPR
F      R/O     10579  101   FONTS
I      R/O       877  104   GDDM
S      R/O       712  190   MNT190
X/X    R/O       828  19B   TOOLS
Y/S    R/O       992  19E   MNT19E
Z      R/W        28  DIR   VMSYS:KRIS.GUIMON
Ready;
```

2.6.2 A BFS File Space

With BFS, file spaces still exist and are also created with the ENROLL command. For example:

```
ENROLL USER ROOT VMSYS (BLOCKS 10000 BFS
```

This file space can only be used after "mounting," for example:

```
OPENVM MOUNT /../VMBFS:VMSYS:ROOT/ /
```

From then on all files and directories stored in the ROOT file space can be used. Unlike CMS, one does not use an ACCESS command to define the search order (a PATH definition is used). And, normally it is not only the end-user named "ROOT" that uses the files and directories: all POSIX users use the ROOT file space.

In POSIX some directory naming conventions[1] exist: directory "/bin" for example contains executables and "/home" hosts the directories for end-user files. In our system one can find:

```
/home/bengt/...
/home/dave/...
/home/kris/...
/home/sal/...
```

End-users then create their own directories under "/home/*userid*/."

Remember: Unless using tricks (such as symbolic links), the space for BFS end-users is all located in file space ROOT.

2.6.3 Combining File Spaces : SFS Aliases - BFS Links

In an SFS, one can use **aliases** to assign multiple names to a given file or to make the same file visible in multiple directories or file spaces. For example:

```
CREATE ALIAS REXXTRY EXEC VMSYS:KRIS.MYEXECS REXXTRY XEDIT VMSYS:KRIS.XEDITS
```

In BFS, one can create **links**, **symbolic links**, and **external links** to define alias-like objects. But links are more powerful than SFS aliases: with a link even a directory (and everything below it) can be made visible at other places. For example, when the GUIMON sample application is located in directories below "/home/kris/guimon," KRIS and SAL would see it as their own after issuing:

```
OPENVM CREATE SYMLINK /home/kris/guimon /home/sal/guimon
```

This would create the following structure in "/home"

```
/home
/home/kris
/home/kris/guimon
/home/kris/guimon/client
/home/kris/guimon/server
/home/sal
/home/sal/guimon           -> /home/kris/guimon
/home/sal/jnrCMS
```

To SAL, however, his "/home" directory would simply look like:

[1] More information about the conventions can be found in Figure 7 on page 10.

```
/home
/home/sal/guimon
/home/sal/guimon/client
/home/sal/guimon/server
/home/sal/jnrCMS
```

It is even possible to create links in one BFS to objects residing in another BFS server. For example, suppose that “/home/kris” is located in the ROOT file space of VMSYS, but “/home/bengt” in TESTROOT of VMSYSU.

```
OPENVM UNMOUNT /
OPENVM MOUNT /../VMBFS:VMSYSU:TESTROOT/ /
OPENVM CREATE SYMLINK /home/bengt/guimon /../VMBFS:VMSYS:ROOT/home/kris/guimon
```

Links are very practical and heavily used by the VM Java and NetRexx installation process: Java and NetRexx are installed in separate BFS file spaces, but symbolic links allow one to find everything as if it were installed in the ROOT file space.

After installation, the ROOT file space in the BFS could look like:

```
/
/dev
/etc
/home
/home/kris
/home/kris/guimon
/home/kris/guimon/client
/home/kris/guimon/server
/home/sal
/home/sal/guimon          -> /home/kris/guimon
/home/sal/jnrCMS
/lib                      -> /usr/lib
/opt
/usr
/usr/bin
/usr/include
/usr/include/sys
/usr/java                 -> /../VMBFS:VMSYS:_JAVA/./J1.1/
/usr/lib
/usr/lib/nls
/usr/lib/nls/msg
/usr/lib/nls/msg/C.IBM-1047
/usr/lib/nls/msg/En_US.IBM-1047
/usr/lpp
/var/spool/mail
/usr/openvm
/usr/openvm/java          -> /../VMBFS:VMSYS:_OVMJAVA/java
/usr/openvm/NetRexx       -> /../VMBFS:VMSYS:_NETREXX/NetRexx
/usr/pub
/usr/NetRexx              -> /../VMBFS:VMSYS:_NETREXX/./NetRexx
```

From the above, one can conclude that Java is installed in a BFS file space _JAVA, but a link in ROOT's “usr” directory makes it visible to all users mounting ROOT.

After the installation of the Java packages we also issued a QUERY LIMITS command to see what file spaces exist and how many 4K blocks are used.

```
q limits  all
Userid    Storage Group  4K Block Limit  4K Blocks Committed  Threshold
_JAVA             2               12000            7322-61%      100%
_NETREXX          2                1000             425-42%      100%
_OVMJAVA          2                 125              57-45%      100%
KRIS              2               10000            2019-20%       90%
MAINT             2                   0               0-00%       90%
MAINT220          2              100000           82838-82%       90%
ROOT              2               10000            2535-25%      100%
RTMESA            2                1000             209-20%       90%
TMP               2                1000               0-00%      100%
VAR               2                1000               0-00%      100%
Ready; T=0.01/0.01 18:42:54
```

2.7 Installing Java and NetRexx without the Shell and Utilities

At the time of writing this book, the VM/ESA platform′s port of the Java Developer Kit, as developed to run with the OpenEdition Shell and Utilities feature, is proceeding through the Java Compatible test process. It will be made Generally Available ("GA") and will carry the Java Compatible logo once it is proven to pass the Java Compatible test suite.

An alternative execution environment, created by the VM platform developers in Endicott together with the authors of this redbook in Böblingen, removes the need for customers to purchase the OpenEdition Shell and Utilities feature by offering a "shell-less" Java implementation. As this environment has not yet been verified as Java Compatible, it is offered as a "Beta" release only.

The following steps will guide you through the installation of Java and NetRexx without the OpenEdition Shell and Utilities feature being installed on your system.

Before you begin to install Java and NetRexx you must have access to the Byte File System. If you do not have a file pool installed on your base VM/ESA system, please see section 2.2, "Overview of the SFS" on page 3 and the manual *CMS File Pool Planning, Administration, and Operation*, SC24-5751, for more information on how to build a file pool.

It is useful to know that Java and NetRexx are installed using LOADBFS control files.

JAVA LOADBFS This file controls the installation of Java: a BFS user is enrolled, two TAR files are unpacked and links are created in ROOT.

NETREXX LOADBFS This file controls the installation of NetRexx: a BFS user is enrolled, one TAR files is unpacked and links are created in ROOT.

You can tailor these LOADBFS files, for example to change the filepool to your needs. Detailed tailoring instructions are provided in the files. To install using these LOADBFS files, the POSIX Shell and Utilities are not required.

Warning: You may have to increase the virtual machine size of MAINT to ***200MB*** because of the size of one of the files you will use.

2.7.1 Major Steps to Install "Shell-less" Java and NetRexx

1. Obtain the VM/ESA Java and NetRexx code
2. Set up the Java and NetRexx environment
3. Build Java and NetRexx
4. Download and install the OVMJAVA package
5. Download and install the SG245148 package

2.7.1.1 Obtain the VM/ESA Java and NetRexx Code

Complete information on how to obtain the Java and NetRexx code for VM/ESA may be found on the Web, starting from the VM Java home page at URL `http://www.vm.ibm.com/java/`. There are two choices:

1. You may download the Java and NetRexx code directly from the Web by following the Downloads link from the VM Java home page.
2. You may order the Java and NetRexx code as service, using the PTF numbers that may be found by following the Service link from the VM Java home page. (As we go to press, the current PTF numbers are UM28908 for Java and UM28789 for NetRexx.)

Note: In either case, you must also ensure that CMS PTFs UM28787 and UM28862 have been applied since they provide necessary CMS support for the JVM.

2.7.1.2 Set up the Java and NetRexx Environment

1 Log on to the MAINT user ID.

ENTER logon maint

Ready; T=n.nn/n.nn hh:mm:ss

2 Set up the Java and NetRexx environment.

Approximately 80 cylinders of 3380 DASD, or its equivalent, is required. If you do not have this much free space available already, then allocate it now before any build steps.

a Determine if you need to allocate additional space.

query filepool minidisk vmsys:

Examine the allocated size of storage group 2 and use this information to determine how much to increase the filepool space to accommodate the 12,000 4K blocks needed for Java and NetRexx.

b If needed, add minidisks to storage group 2 in the filepool VMSYS. (The user ID for VMSYS is VMSERVS.) See *CMS Filepool Planning, Administration and Operation*, SC24-5751, for more information on adding space to a storage group.

3 Create the BFS file space for ROOT.

Creating the basic BFS setup can be done by issuing the `LOADBFS BFS` command. Note that the following entries are required in the CP directory.

```
DIRECTORY ...
 GLOBALDEFS
  ⋮
  POSIXGROUP system   0
  POSIXGROUP staff    1
  POSIXGROUP bin      2
  POSIXGROUP sys      3
  POSIXGROUP adm      4
  POSIXGROUP mail     6
  POSIXGROUP security 7
  POSIXGROUP nobody   4294967294
  ⋮
***************************
* Userids for Posix
***************************
USER ROOT NOLOG 32M 32M G
 POSIXINFO UID 0 GNAME system
USER DAEMON NOLOG 32M 32M G
 POSIXINFO UID 1 GNAME staff
USER BIN NOLOG 32M 32M G
 POSIXINFO UID 2 GNAME bin
USER SYS NOLOG 32M 32M G
 POSIXINFO UID 3 GNAME sys
USER ADM NOLOG 32M 32M G
 POSIXINFO UID 4 GNAME adm
USER NOBODY NOLOG 32M 32M G
 POSIXINFO UID 4294967294 GNAME nobody
USER DEFAULT NOLOG 32M 32M G
 POSIXINFO UID 4294967294 GNAME DEFAULT
```

If you plan to enroll ROOT into a Byte File System other than VMSYS, you have two choices. You may modify the BFS LOADBFS file on MAINT 193 appropriately before running LOADBFS BFS. Alternatively, you may follow the procedure outlined below, referencing your file pool name instead of VMSYS.

a Enroll user ROOT into VMSYS.

enroll user root vmsys: (blocks 1000 storgroup 2 bfs

b Mount the Byte File System.

Note: All commands from this point are case sensitive.

openvm mount /../VMBFS:VMSYS:ROOT/ /

c Create directory usr

openvm create directory /usr

d Create directory home

openvm create directory /home

The directory home is not really needed for the installation, but might be a good place to put your own tools.

2.7.1.3 Build Java and NetRexx

If you plan to install Java and NetRexx into a Byte File System other than VMSYS, you need to update JAVA LOADBFS and NetRexx LOADBFS to specify the file pool you wish to use. You should replace all VMSYS references with your file pool name.

4 Build Java

vmfbld ppf esa cmsprod dmsbljav (all setup

DMSBLJAV is the name of the build list for Java.

5 Build NetRexx

vmfbld ppf esa cmsprod dmsblnrx (all setup

DMSBLNRX is the name of the build list for NetRexx.

2.7.1.4 Download and Install the OVMJAVA Package

The OVMJAVA package provides an interface for executing Java commands that does not require prior installation of the Open Edition VM/ESA Shell and Utilities feature. The components of OVMJAVA are organized as follows:

- The /usr/openvm/java/bin directory created by the installation of this package contains CMS modules which allow calls to JDK tools using the openvm run command.
- The /usr/openvm/java directory contains sample REXX execs which can be copied to a CMS minidisk or SFS directory for use.
- The /usr/openvm directory ties together Java, NetRexx, and the SG245148 package described below (see 2.7.1.5, "Download and Install the SG245148 Package") to allow the use of the JC, NRC, and NRR commands developed during the residency (see 3.5, "JC EXEC - Java Compile" on page 31, 3.4, "NetRexx Compile" on page 29, and 3.6, "NetRexx Run" on page 31, respectively, for complete descriptions).

The default CLASSPATH includes the individual CLASSPATH entries required to use these three packages as well as the current working directory. Additional search elements can be appended to the start of the CLASSPATH search list using the SETCENV command (described in 3.8, "SETCENV - Setting C Environment Variables" on page 33).

The OVMJAVA package may be obtained from the VM Download Library, located on the World Wide Web at URL `http://www.vm.ibm.com/download/`. Complete instructions for downloading packages from the library are provided there; specific instructions for installing OVMJAVA are included in the OVMJAVA package.

2.7.1.5 Download and Install the SG245148 Package

All of the programs developed during the residency and described in this redbook are also available in the VM/ESA Download Library, located on the World Wide Web at URL `http://www.vm.ibm.com/download/`, in the SG245148 package. Included, as you will discover as you continue your reading of this redbook, are an extensive collection of Java and NetRexx examples, featuring a package of CMS utility classes and a complete, sample distributed application, as well as a set of program development tools that were critical to our productivity during the residency.

Complete instructions for downloading packages may be found on the initial Download Library web page. Installation instructions for the various sample program packages are included with the descriptions of the programs that appear elsewhere in this redbook.

2.8 Adding a NetRexx Developer User ID

Because authorizations are different in a POSIX environment, each CMS user ID may need a *uid* and *gid* added to its CP user directory entry. In addition you may want to establish a default mount point and initial working directory. All of these options are defined by using the *POSIXINFO* Directory Control Statement.

You should create a BFS file space for each user to develop NetRexx applications, or a file space to be shared by a department. Use the CMS ENROLL command with the BFS parameter to enroll the user ID in the file pool of your choice.

To use these additional file spaces, external links or additional mounts will be required.

To attach a user BFS file space to a directory tree so that the user file space is implicitly mounted when the user enters his or her home directory, use the OPENVM CREATE EXTLINK facility. For example:

```
/* Create a mount external link */
openvm create extlink /../VMBFS:VMSYSU:ROOT/home/cmsuser ,
        MOUNT /../VMBFS:VMSYSU:CMSUSER/
```

Finally, you need to be aware of POSIX security issues. What kind of file permission should users have? Should any one with access to BFS have access to your NetRexx application?

Chapter 3. Tools Used During the Project

Regular CMS users may find the OPENVM SHELL a strange and confusing environment. Given the demands imposed by the business, taking the time to learn a new command environment may not be an option. Also the cost of the VM/ESA OpenEdition Shell and Utilities feature may be not an option for some VM installations.

During the project several tools were created to make the life of a CMS NetRexx and Java programmer easier. They provide the basic capabilities needed to develop and run Java and NetRexx programs. These tools are included in the sample program set for this book. See 2.7.1.5, “Download and Install the SG245148 Package” on page 19 for information on obtaining the entire package of sample programs for your VM/ESA system.

The whole environment centers around the XEDIT, BFSLIST, NRC, and NRR commands.

3.1 XEDIT

XEDIT allows you to create and modify BFS files. It works just as in CMS, but you need a special parameter NAMETYPE BFS. To create or modify a file, simply type its BFS pathname, for example:

```
xedit /home/steve/myFirstProgram.nrx (nametype bfs
```

If you have the correct CP directory statement to set your current directory you do not need to specify your home directory explicitly. You can also set the current working directory with the OPENVM SET DIRECTORY command.

3.1.1 PROFILE XEDIT

Many programmers will have to modify their PROFILE XEDIT to make it work for BFS files.

First of all, the PROFILE XEDIT must be written in REXX to be able to edit BFS files.

Some XEDIT profiles analyze the fileid for which XEDIT is being called, for example:

```
Parse Arg fn ft fm '(' options
```

However, when a user issues “x AboutFrame.nrx (nametype bfs,” the Parse Arg shown above yields:

```
fn = "ABOUTFRA"
ft = ""
fm = ""
options="        NAMETYPE BFS     "
```

So there is no variable holding the BFS requested fileid.

As BFS fileids can even contain spaces or parentheses, a Parse Arg as shown above cannot easily be used to extract the fileid. Therefore XEDIT passes the BFS fileid as arg(2) to PROFILE XEDIT.

© Copyright IBM Corp. 1998

A simple solution might be:

```
BFS=(arg()>1)          /* Is this is a BFS file ? */
if bfs then parse arg '(' options , fn '' ft fm
       else parse arg fn ft fm '(' options
 :
'COMMAND LOAD' fn ft fm '(' options
 :
```

When the PROFILE XEDIT wants to do some setup depending on the filetype, a few changes are required as well. An example analyzing a BFS fileid:

```
'COMMAND EXTRACT /FN/FT/FM/PNAME/'
if pname.2<> '' then do /* This is a BFS file */
   Parse Value Reverse(pname.2) with ext '.' +0 name '/' +0 Path
   Path = Reverse(Path)
   name = Reverse(name)
   type = Reverse(ext)
   end
else type=ftype.1
 :
```

3.2 BFSLIST - Listing the Contents of a BFS Directory

As mentioned earlier, Java programs are stored in the BFS. As FILELIST cannot be used to list BFS files, you feel lost, you cannot easily see what files and directories are available. There is the OPENVM LISTFILE command and the ls POSIX Shell command. But as illustrated below, they produce line mode output, hence don't compare with FILELIST but with LISTFILE. Note also that both OPENVM LISTFILE and the shell command ls -al return a GMT time as time of last change whereas we are used to seeing local times.

3.2.1 OPENVM LISTFILE

Below you can find the console output of a few OPENVM commands. Before using a BFS, a file system must be mounted. In a native CMS environment this is done by the OPENVM MOUNT command. Then we issue OPENVM SET DIR to define our working directory.

Readers with a Unix background are used to the above terminology. Readers with CMS experience may need some help: one could compare OPENVM MOUNT with CP LINK someuser somedev 191 M or with SET FILEPOOL mypool. Whereas OPENVM SET DIR can be compared accessing the "A-disk": ACCESS somedev A or ACCESS someuser.somesubdir A.

Note: The BFS is a case sensitive file repository (all of Unix is case sensitive: ls is not the same as Ls). Therefore all parameters of the OPENVM command that refer to files are case sensitive too: /home/kris exists in BFS, but /home/KRIS doesn't.

```
openvm mount /../VMBFS:VMSYS:ROOT/ /
Ready; T=0.05/0.06 10:12:01

openvm set dir /home/kris
Ready; T=0.01/0.01 10:12:38

openvm q dir
Directory = '/home/kris'
Ready; T=0.01/0.01 10:13:12

openvm listfile
Directory = '/home/kris'
Update-Dt  Update-Tm Type  Links          Bytes Path name component
02/19/1998 19:10:24   F        1            444 '.profile'
02/17/1998 09:34:57   F        1            954 'abc.class'
02/17/1998 09:34:48   F        1            106 'abc.crossref'
02/17/1998 13:50:16   F        1            410 'abc.nrx'
02/03/1998 19:52:05   E1       -              - 'cmsddr'
02/04/1998 16:30:46   E1       -              - 'cmspipe'
02/19/1998 08:41:45   F        1           1003 'getProperties.class'
02/19/1998 08:41:36   F        1            114 'getProperties.crossref'
02/19/1998 08:41:58   F        1            129 'getProperties.nrx'
02/04/1998 16:09:42   F        1           2097 'kris2.class'
02/04/1998 16:09:32   F        1             85 'kris2.crossref'
02/04/1998 15:59:23   F        1            216 'kris2.nrx'
02/03/1998 16:48:45   F        1              5 'kris3.nrx'
02/19/1998 11:06:30   F        1           2675 'myCanvas.class'
02/24/1998 10:40:33   F        1            654 'myCloseMenu.class'
02/24/1998 10:40:33   F        1           3819 'myEventClass.class'
02/24/1998 10:40:33   F        1            634 'myFrameController.class'
02/24/1998 12:32:00   D        -              - 'testDirectory'
02/24/1998 09:40:33   L        -              - 'testLink21'
⋮
```

Automatic Mount

Helpful Hint

When becoming a regular BFS user, you might want to make your user "BFS ready" automatically. This may be done either by updating the POSIXINFO statement in the CP directory, for example:

```
POSIXINFO UID 0 GNAME staff IWDIR /home/kris/,
          FSROOT '/../VMBFS:VMSYS:ROOT/'
```

or by adding two calls to OPENVM to your PROFILE EXEC, such as:

```
'EXEC OPENVM Mount /../VMBFS:VMSYS:ROOT/ /'
'EXEC OPENVM Set Dir /home/kris'
```

3.2.2 POSIX Shell and Utilities

With the POSIX Shell and Utilities feature of VM, one gets the common Unix commands, such as `ls`, `mkdir` and `mount`. The POSIX Shell is started by entering `POSIX` on the command line.

Most CMS users have a `PROFILE EXEC` that is run to personalize their CMS environment. In the POSIX Shell a `.profile` file performs a similar function for the Unix environment: a filesystem is mounted and a working directory is defined. So for brevity we don't show `mount` or `cd` commands in the console that follows.

```
posix

Licensed Material - Property of IBM
5654-030 (C) Copyright IBM Corp. 1995
(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.

kris #
pwd
/home/kris
kris #

ls -al
total 952
-rwxrwxrwx   1 kris     system      444 Feb 19 18:10 .profile
-rw-rw-r--   1 kris     system     7398 Feb  6 13:48 ClntSock.class
-rw-rw-r--   1 kris     system      550 Feb  6 13:48 ClntSock.crossref
-rw-rw-r--   1 kris     system     3501 Feb  6 10:29 ClntSock.java.keep
-rw-rw-rw-   2 kris     system     1716 Feb 10 10:58 ClntSock.nrx
-rw-rw-rw-   1 kris     system      744 Feb 16 12:47 Game.nrx
-rw-rw-rw-   1 kris     system     1165 Feb 16 12:46 Game2.nrx
-rw-rw-r--   1 kris     system     1620 Feb 16 14:08 Game3.class
-rw-rw-r--   1 kris     system      178 Feb 16 14:07 Game3.crossref
-rw-rw-rw-   1 kris     system      821 Feb 16 13:20 Game3.nrx
-rw-rw-rw-   1 kris     system      875 Feb  6 15:46 GetHead.nrx
-rw-rw-r--   1 kris     system    14173 Feb 24 12:51 GetPerf.class
-rw-rw-r--   1 kris     system     1425 Feb 24 12:50 GetPerf.crossref
drwxrwxrwx   1 kris     system        0 Feb 24 12:32 testDirectory
lrwxrwxrwx   1 kris     system       12 Feb 24 09:40 testLink21 -> test/test21/
⋮
```

3.2.3 BFSLIST

Being spoiled with fullscreen tools such as FILELIST and DIRLIST, having to work with linemode only tools feels like returning to the stone age. In addition, when BFS, with Unix file structures, is new to you, you will feel even more lost. Only when you start getting an overview of the BFS directories and files will you begin to feel at home.

In a previous project a fullscreen tool, named BFSLIST was created to provide CMS users with a more user-friendly interface to the BFS. It is offered with redbook *OpenEdition for VM/ESA Implementation and Administration*, SG24-4747. That version of BFSLIST had some shortcomings and did no longer work perfectly in VM/ESA 2.3.0. So we greatly enhanced it.

The new version of BFSLIST is pictured below. The end-user interface is now more like FILELIST: you can enter commands against the listed BFS objects.

```
 KRIS  BFSLIST  A0  V 1000  Trunc=1000 Size=94 Line=1 Col=1 Alt=0
DIRECTORY: /home/kris/
--> cmd:  X(Xedit/list) A(asciiXedit) D(del) R(ren) C(copy) Q(qLink) T(tree)
cmd T Fileid            Privs     Owner    Group       Size Date       Time
    F GetPerf.class     rw-rw-r-- kris     system       13K 1998/02/24 12:51:20
    F GetPerf.crossref  rw-rw-r-- kris     system        1K 1998/02/24 12:50:56
    F GetPerf.nrx       rw-rw-rw- kris     system        4K 1998/02/24 12:50:08
    F TimeFrame.class   rw-rw-r-- kris     system        9K 1998/02/24 12:32:01
    F TimeFrameAction.c rw-rw-r-- kris     system        2K 1998/02/24 12:32:01
    F TimeFrameClose.cl rw-rw-r-- kris     system      665  1998/02/24 12:32:01
    F TimeFrameControll rw-rw-r-- kris     system      645  1998/02/24 12:32:01
    F TimeFrameEvents.c rw-rw-r-- kris     system        2K 1998/02/24 12:32:01
    D testDirectory     rwxrwxrwx kris     system        0  1998/02/24 12:32:00
    l testLink21        rwxrwxrwx kris     system        0  1998/02/24 12:30:08
    F GuiMon.nrx        rw-rw-rw- kris     system        9K 1998/02/24 11:12:42
    F myCloseMenu.class rw-rw-r-- kris     system      654  1998/02/24 09:40:33
    F myEventClass.clas rw-rw-r-- kris     system        3K 1998/02/24 09:40:33
    F myFrameController rw-rw-r-- kris     system      634  1998/02/24 09:40:33
    F myMenuAction.clas rw-rw-r-- kris     system       10K 1998/02/24 09:40:33
    F TestFrame.class   rw-rw-r-- kris     system       11K 1998/02/24 09:40:33
    F GuiMon.class      rw-rw-r-- kris     system      978  1998/02/24 09:40:32
    F GuiMon.crossref   rw-rw-r-- kris     system        2K 1998/02/24 09:39:55
    F GuiMon.java.keep  rw-rw-r-- kris     system       31K 1998/02/24 09:30:53
    F TimeGraph.class   rw-rw-r-- kris     system        3K 1998/02/24 08:46:28
    F TimeGraph.java    rw-rw-rw- kris     system        4K 1998/02/24 08:44:54
    F GuiMonVM.class    rw-rw-r-- kris     system        3K 1998/02/23 19:50:50
    F GuiMonVM.crossref rw-rw-r-- kris     system      749  1998/02/23 19:50:38
    F GuiMonVM.nrx      rw-rw-rw- kris     system        2K 1998/02/23 19:49:57
    F GuiMon-1.nrx      rw-rw-rw- kris     system        7K 1998/02/20 17:17:49
1= Help       2= Refresh  3= Quit   4= Sort(NAME)  5= All        6= Sort(SIZE)
7= Backward   8= Forward  9= Dirs  10= Parent     11= XEDIT/List 12= All(.nrx)
```

A few usage notes are appropriate.

3.2.3.1 BFS File Naming - BFSLIST Terminology

BFSLIST uses the terms *filename* and *filetype*. They do have a slightly different meaning than in a traditional CMS environment. Let's take the third file shown by BFSLIST as an example:

```
  F GetPerf.nrx       rw-rw-rw- kris     system        9K 1998/02/24 12:50:08
```

The full name of this file is /home/kris/GetPerf.nrx, which is called a ***pathname*** in the POSIX environment (in CMS you can issue HELP OPENVM PATHNAME to learn more about pathnames).

In this pathname we can distinguish a *path*: /home/kris/ and a *name*: GuiPerf.nrx. CMS users may tend to say: "*GetPerf* is the filename and *nrx* is the filetype." This seems OK as there are two distinct parts in the name, just as in traditional CMS file names.

But remember that /home/kris/GuiMon.java.keep is a valid BFS file name too. As it has three parts in the name, the CMS filename and filetype terms are no longer appropriate.

On the other hand, a BFS holds different types of objects: files, directories, links, symbolic links and external links (issue HELP OPENVM MENU to learn more about links).

Therefore, in BFSLIST we use *filetype* to refer to the type of object. The filetype is the first word of each BFSLIST line. You can expect to see the following types:

F stands for a file
D stands for a directory
d stands for a symbolic link or an external MOUNT
l (lowercase l) stands for a link
? is an external link (for example to a CMS file)
others issue HELP PIPE BFSSTATE to find a description for other types.

You can use BFSLIST's Q command to find more information about link objects.

3.2.3.2 Execution Parameters

BFSLIST accepts one parameter: the pathname to list. The default is your current working directory, as defined in the POSIX Shell or with the OPENVM SET DIR command.

To list the root directory, issue BFSLIST /.

3.2.3.3 Walking Through the Directory Tree

PF9 can be used to only see directories, and PF11 will open the directory pointed to with the cursor. Use PF3 to close a directory and PF5 to see all entries of the listed directory.

PF10, *Parent*, opens the parent directory of the directory being listed.

3.2.3.4 Sorting

The list of objects can be sorted, either by entering sort commands in the command line, or by using a predefined PF key. The following sort commands are defined: SNAME, SDATE, SSIZE, and STYPE.

3.2.3.5 Commands

With the current version of BFSLIST you still cannot execute any arbitrary user command. Only commands defined in $BFSEXEC XEDIT can be executed. The most important commands are listed on top of the screen. You will be happy to find the most common ones:

X to XEDIT a file or list a directory (this is also PF11)

A to XEDIT an ASCII encoded file (this is also PF23; uses the ASCXED EXEC)

C copy a file (you will be prompted for the new name)

D delete a file or directory (you'll be asked for confirmation)

R rename a file or directory (you will be prompted for the new name)

L create a LINK to a file or a directory (a prompt for the name follows)

Q to query a LINK (or for a FILE: show the complete name)

T to start BFSTREE

= to execute the same command as before

? to retrieve the last saved command

/ makes this line the top line of BFSLIST

GET copies a BFS file onto a CMS minidisk or SFS directory

NRC executes the NetRexx compiler (NetRexx Compile)

NRR executes a NetRexx class file (NetRexx Run)

JC executes the Java compiler (Java Compile)

JR executes a class file (Java Run)

3.2.3.6 Installing

Installing BFSLIST is nothing special: copy the files to a public minidisk or directory. This is a list of the required files:

```
BFSLIST  EXEC     =   Starts a BFSLIST
$BFSLIST XEDIT    =   The "PROFILE" and "lister" for BFSLIST
$BFSEXEC XEDIT    =   The "EXECUTE" for BFSLIST and BFSTREE
BFSLIST  HELPCMS  =   Some online help
```

As the files are placed in a PACKAGE, it is also possible to execute:

```
FILEList BFSLIST PACKAGE (Filelist
```

3.2.3.7 Shortcomings

BFSLIST has several shortcomings:

- You can only execute predefined commands. Entering options for these commands is impossible too. So entering a FILELIST-like command as RENAME / newname is not supported.
- The visible part of the lines is not refreshed after the execution of a command. Use PF2 to refresh the whole list.
- BFSLIST has partial support for pathnames that use the special characters: blank, ′, ″, (,), *, and =. The XEDIT, COPY, RENAME and DELETE commands will work, others may fail.
- BFSLIST does not accept wildcards. BFSLIST /home/kris/*.nrx for example will simply yield an error message: *File or directory does not exist*. As a bypass: use XEDIT′s ALL command while in BFSLIST, for example ALL /.nrx/.

3.2.3.8 Tailoring

As opposed to FILELIST, there is no separate PROFILE macro. So users cannot easily tailor their own PF keys, colors, or other preferences.

The good news though is that BFSLIST is based on classic tools: REXX, CMS Pipelines (BFSDIRECT and BFSSTATE), and a few calls to the OPENVM command.

BFSLIST EXEC This exec does almost nothing, it starts $BFSLIST XEDIT.

$BFSLIST XEDIT This is the exec that creates the list, and defines the PF keys, colors and so on. Change the "Profile:" routine to tailor these attributes.

$BFSEXEC XEDIT This macro executes the commands you enter in BFSLIST and BFSTREE. Adding your own commands can be done in routine "Execute."

3.3 BFSTREE - Listing a BFS Directory Tree

BFSTREE is a bit similar to BFSLIST, but it lists all directories found from a given starting point.

Warning

Beware: BFSTREE can **consume significant system resources** especially for the command BFSTREE / as then you request BFSTREE to scan the whole BFS space starting at the ROOT directory. The BFS server seems to have to do a lot of work with that. Using BFSLIST / is not that costly.

Note as well that while BFSTREE is running, the BFS server holds locks for you; the longer these locks are held, the more other people may become annoyed with you.

The display produced by BFSTREE /home/ is printed below:

```
 KRIS  BFSTREE  A0  V 1000  Trunc=1000 Size=139 Line=0 Col=1 Alt=2
 Viewing tree starting at: /home/

   /home/
    bengt
    dave
    java
     classes
      java
       applet
       awt
        datatransfer
        event
        image
        peer
       .......... some lines suppressed .....
    kris
     test
      test21
       test31
       test32
        test4
       test33
      test22
     testLink21 -> test/test21/
     test1
    sal
     client
      v1
1= Help       2= Refresh  3= Quit   4=            5=            6=
7= Backward   8= Forward  9=       10= Parent     11= List       12=
====>
```

Note that BFSTREE shows directories that are links or symbolic links as:

```
linkpath -> realpath
```

BFSTREE could have scanned for all directories in that link too, but that effort would even be more computing resource intensive. You can enter the T or the X command to list the link's contents.

3.3.1.1 Commands

The commands supported by BFSTREE are a subset of those of BFSLIST. Only commands defined in $BFSEXEC XEDIT can be executed.

X to list a directory (this is also PF11)

D delete a directory (you'll be asked for confirmation)

R rename a directory (you will be prompted for the new name)

L create a LINK to the a directory (a prompt for the name follows)

T to start BFSTREE on the directory

= to execute the same command as before

/ makes this line the top line of BFSLIST

? to retrieve the last saved command

3.3.1.2 Tailoring

BFSTREE is similar to BFSLIST in this area too: users cannot easily tailor their own PF keys, colors, or other attributes.

The following REXX procedures are used:

BFSTREE EXEC This exec does all the work, including the definition of the PF keys, colors and so on. Change the “XEDIT:” routine to tailor these attributes.

$BFSEXEC XEDIT This macro executes the commands you enter in BFSTREE and BFSLIST. Adding your own commands can be done in routine “Execute.”

3.4 NetRexx Compile

Obviously during the project a lot of NetRexx compiles were started as you might expect. Two procedures helped us here.

3.4.1 NRC EXEC - NetRexx Compile

The NRC EXEC has a few nice advantages:

- The parameters for the NetRexx compiler can be passed in a VM fashion and abbreviations are permitted, for example:

  ```
  NRC myTestNet.nrx (Keep strictc
  ```

- The current directory is temporarily set to the location of the NetRexx source. This is required when wanting to get the compiler output stored in the same directory as the NetRexx source.
- We verify that the LE/370 runtime library (SCEERUN LOADLIB) is in the global loadlib list. If not, the GLOBAL LOADLIB list is updated.
- The NetRexx compiler is a Java program also, and quite an amount of virtual storage is required during compilation. The NRC EXEC specifies 5MB in the Java option (refer to A.9, “Virtual Storage Requirements” on page 143 for more information about virtual storage requirements).
- Another option we pass to Java is the wanted classpath (refer to 3.8.1, “Important Environment Variables” on page 33 for more information about classpath).
- After a successful compilation, the compiled program is started (unless the NORUN option is specified).

The format to start the NRC EXEC is as follows:

```
NRC  pgmname < ( <options>  <PARMS execution parms> >
```

Where

pgmname is the, case sensitive, name of the NetRexx program to compile.

options are the options for the NetRexx compiler or for the NRC EXEC. The recognized options are:

```
  NORUN      tells not to run the compiled program
  -xxxx      any option starting with a -sign is passed untouched
             to the NetRexx compiler.
  PARMS xxx  everything following PARMS is passed untouched
             to the compiled program when it is started.
The following runtime NetRexx options are recognized, the minimum
abbreviation is shown in uppercase.
  Binary       Crossref      COMpile      Diag          Explicit
  Format       Keep          Logo         Replace       SOUrcedir
  STRICTARgs   STRICTASsign  STRICTCase   STRICTSignal  Time
  Trace        Utf8
```

Note: The NetRexx language has an OPTION statement by which you can define most of the above compile options. The options entered on the OPTION statement overrule those specified at runtime.

The result of the compilation are one or more "class" files. The name of a class file ends with **.class** and the NRC EXEC stores them in the same directory as the NetRexx source. For example:

```
NRC /home/kris/test/myServer.nrx
```

will generate (at least) the following file:

```
/home/kris/test/myServer.class
```

3.4.2 NRC XEDIT - NetRexx Compile

What does a NetRexx programmer want to do after using XEDIT to apply some changes to the program source? That's right, compile it! Therefore, we also created NRC XEDIT.

NRC XEDIT will call the NRC EXEC, passing the name of the program being edited.

The format to start NRC XEDIT is as follows:

```
NRC  <any option understood by NRC EXEC>
```

As an example, suppose you are editing myTestNet.nrx, and want to compile it with the KEEP option, just enter nrc keep in XEDIT's command line.

An extra advantage, not found in the NRC EXEC, is that you can define "standard" options for the NRC EXEC. Include one or two special comment lines in the first 20 source lines, starting in column 1:

```
--NRC-XEDIT-OPTIONS  non-overridable options for NRC
--NRC-XEDIT-DEFAULTS overridable options for NRC
```

This is not the same as using the NetRexx OPTION statement because:

- the NetRexx options "nocompile," "keep," and "time" cannot be entered on the OPTION statement.
- the NORUN and PARMS options are understood only by the NRC EXEC.

So our options cards do complement NetRexx's OPTION statement. The differences between our two cards are:

- When NRC XEDIT finds a --NRC-XEDIT-OPTIONS card, all options on that card are passed to the NRC EXEC. The major use probably is for NORUN. For

example a NetRexx program that uses the AWT class cannot run on VM. You would include a --NRC-XEDIT-OPTIONS NORUN card in its source.

- The --NRC-XEDIT-DEFAULTS card is only used when you enter NRC without any options. If on the other hand you enter NRC blabla, NRC XEDIT will not even look for a --NRC-XEDIT-DEFAULTS card.

Note: As mentioned above, the NRC EXEC does not scan the source for these option cards.

3.5 JC EXEC - Java Compile

The JC EXEC can be used to compile a Java program. It is less evolved than the NRC EXEC.

The only "special" things it does are:

- The current directory is temporarily set to the location of the Java source. This is required when wanting to get the compiler output stored in the same directory as the Java source.
- We verify that the LE/370 runtime library (SCEERUN LOADLIB) is in the global loadlib list. If not, the GLOBAL LOADLIB list is updated.

3.6 NetRexx Run

3.6.1 NRR EXEC - NetRexx Run

The NRR EXEC can be used to run a NetRexx program. In practice, the NRR EXEC can start any Java program. Obviously, to run a NetRexx program it must have been previously compiled, meaning that some .class files must exist.

The official way to have the Java virtual machine start a .class file is:

```
java <runtime options> classfile <program parameters>
```

An example:

```
java myServer
```

Java looks for the .class files only in what has been defined as **classpath**. Most users will include the "current directory" in their classpath (refer to 3.8.1, "Important Environment Variables" on page 33 for more information about classpath).

The NRR EXEC makes it a bit easier to run class files stored somewhere else.

The format to start NRR EXEC is as follows:

```
NRR fileid <any input for the program>
```

The NRR EXEC splits the fileid in three pieces: the path, the first part of the filename and the remainder. Only the first two are used.

For example, the two important parts taken from

```
NRR /home/kris/test/myServer.class blabla
```

are: /home/kris/test/ and myServer. The NRR EXEC will set the current directory to /home/kris/test and then start Java with java myServer blabla.

As NRR doesn't care about the extension of the file, you can actually enter:

```
NRR /home/kris/test/myServer.nrx blabla
```

3.6.2 NRR XEDIT - NetRexx Run

NRR XEDIT is similar to NRR EXEC with the extra advantage that you don't have to enter the program name.

The command to start NRR XEDIT is:

```
NRR <any input for the program>
```

Note: NRR XEDIT extracts the name of the BFS file being edited and then calls the NRR EXEC. And as NRR doesn't care about the extension of the file, you can actually enter NRR while XEDITing the NetRexx source.

3.7 Tools for the POSIX Shell Users

As we sometimes also used the POSIX Shell, a few basic functions were added to our .profile files. We also defined a classpath reflecting our environment. Here is our .profile file.

```
PS1='$LOGNAME':'$PWD':'>'
export PS1
NRX='/usr/NetRexx'
export CLASSPATH=/home/java/classes:$NRX/lib/NetRexxC.zip:$NRX/bin:.
alias x="xedit"
alias dir="ls -al"
function nrc {
  java -mx5m COM.ibm.netrexx.process.NetRexxC $1
  return
  }
function nrr {
  java $1
  return
  }
function bfslist {
  cms "BFSLIST" $1
  return
  }
function xedit {
  cms "XEDIT '$1' (NAMETYPE BFS"
  return
  }
```

Notice that thanks to the created functions, it is possible to start a few CMS commands without having to prefix them with "cms." Also, we can type "dir" to list our files. This is the list:

x ... to XEDIT a file. Which is shorter than cms "X ..."
bfslist ... to start BFSLIST
nrc ... to compile a NetRexx program
nrr .. to run a NetRexx program

The NetRexx related functions are not elaborated as we did most of the work from under native CMS.

3.8 SETCENV - Setting C Environment Variables

Unix and PC/DOS environments have so called **environment variables**. In the POSIX Shell one uses the `export` command to define them. For example `export TZ=cet1win` could be used to define the timezone (central european time, one hour offset in this example). The `echo` command can be used to show a value: `echo $TZ` displays `cet1win` in our example.

3.8.1 Important Environment Variables

Some variables are more important than others. We'll list only the most important ones. If you are looking for more information, refer to the Unix documentation or to the redbook *OpenEdition for VM/ESA Implementation and Administration*, SG24-4747, section 2.7, "Environment Variables."

PATH This is definitely the most important environment variable. It defines the directories that the system searches to find programs to execute. Ours was very simple:

```
/bin:/usr/java/bin
```

Notice that a colon (:) separates the different directories, so directory `/bin` is searched, then `/usr/java/bin`. Also notice that the directory names are case sensitive.

CLASSPATH In its turn this is the most important variable for Java users: it defines the directories that Java searches to find class files.

The CLASSPATH we used is as follows:

```
.:/home/java/classes:/usr/NetRexx/lib/NetRexxC.zip:/usr/NetRexx/bin
```

Pay special attention to the period (.) at the beginning. It means that the *current directory* has to be searched before any other directory.

Beware

Don't use the classpath shown in the example above. It is only the classpath we used with VM's beta version of Java. The initial classpath defined during installation can be found in the installation files JAVA LOADBFS and NETREXX LOADBFS, as well as in BFS file /etc/profile. The default classpath for an installation without NetRexx might be

```
usr/java/classes:/usr/java/lib/classes.zip:.
```

Refer to 2.7, "Installing Java and NetRexx without the Shell and Utilities" on page 16 for pointers to further installation information.

Note: Java also has a `-classpath` option. Whereas the CLASSPATH environment variable complements a hardcoded search path, the -classpath option replaces the search path for class files. See 3.8.3, "More About Classpath" on page 34 for an additional discussion of these alternatives.

3.8.2 Setting Environment Variables from CMS

In CMS, the C environment variables can be set using GLOBALV, group CENV. The OPENVM command passes any GLOBALV variable defined in the CENV group to the C environment.

A problem arrives when the value for the variable must be mixed case. When entering

```
Globalv select cenv setp TZ cet1win
```

on the console, CMS translates everything to uppercase. The actual command thus becomes

```
GLOBALV SELECT CENV SETP TZ CET1WIN
```

Whereas uppercasing is no real problem for timezone definition, it is a real problem when defining paths (or programs names). Executing the GLOBALV command from inside a REXX EXEC, which must use **address command**, allows the definition of mixed case values.

We provide a general solution: the SETCENV EXEC. It preserves the case of the value for the environment variable. The format to call SETCENV is:

```
SETCENV varname value
SETCENV Query                (this is default)
SETCENV GETSHELL <varname>
```

Where the parameters mean:

varname Is the name of the environment variable. SETCENV uppercases this name as the C environment variable names are uppercase.

value Is the contents of the environment variable, it starts after the blank that delimits *varname*. The case is preserved. The maximum length is 255, imposed by GLOBALV. SETCENV produces an error message when attempting to define something longer.

QUERY List the environment variables defined in the POSIX Shell profile `/etc/profile` and those defined in GLOBALV.

GETSHELL When no variable name is specified, SETCENV lists all environment variables defined in the POSIX Shell profile `/etc/profile`.

When a variable name is specified, SETCENV extracts the requested environment variable from the POSIX Shell profile `/etc/profile` and defines the same in GLOBALV.

SETCENV saves what you ask in storage and on disk in LASTING GLOBALV A.

3.8.3 More About Classpath

As Java is an interpretive language, the search for a Java program doesn't follow the directories defined in the PATH. Java looks for the environment variable CLASSPATH to find that search path for class files. An alternative is Java's `-classpath` option; it overrides the environment variable.

When neither of these are defined, Java uses a hardcoded search path.

What to choose if the hardcoded path doesn't fit your case?

The POSIX Shell users have two choices to make a permanent change:

- Include an export CLASSPATH=yourstuff in the system profile: /etc/profile. This way the new class path is available to all POSIX Shell users.
- Include an export CLASSPATH=yourstuff in the user's profile: /home/xxxxxx/.profile (where xxxxxx is the user's user ID).

CMS users can also select between two options. But, even though we have a system profile too (SYSPROF EXEC S), it is not common to define GLOBALV variables there. This leaves the next obvious options:

- Most probably, one will have an EXEC to start a Java program. On the call to Java, add the -classpath option; for example, MYJAVA1 EXEC

```
address command   /* Be sure to have case preservation   */
                  /* and not to call EXECs "by accident" */
Appl1Dir ='/JavaApplication1/'
MainClass='MainStuf'
/* do we have the C runtime lib ? */
'PIPE COMMAND QUERY LOADLIB',
      '|SPEC w3-* 1|PAD 5|CHANGE 1.5 /NONE //|VAR libs',
      '|SPLIT|FIND SCEERUN|TAKE|COUNT LINES|VAR HaveC'
if haveC then nop
         else 'GLOBAL LOADLIB SCEERUN' libs

/* Adapt the current directory to the dir of the application */
'PIPE LITERAL CD|BFSQUERY|VAR CD'
'EXEC OPENVM SET DIR' Appl1Dir

Say 'Calling Java program' MainClass'.class ...'
'EXEC OPENVM RUN /usr/java/openvm/java' ,
    '-classpath :/myOtherStuffForJava:'||,
                '/home/java/classes:/usr/NetRexx'||,
    MainClass
src=rc

'EXEC OPENVM Set Dir' cd  /* Restore current directory */
exit(src)
```

- For a user-by-user approach, have the user execute SETCENV CLASSPATH yourstuff once, or have them include the SETCENV call in their PROFILE EXEC:

```
'EXEC SETCENV CLASSPATH .:/MyJavaStuff:...etc....'
```

Chapter 4. Comparing REXX to NetRexx

In this chapter we will try to compare REXX with NetRexx. We hope that at the end readers will be able to position both languages, as well as being able to read NetRexx programs and can code simple programs. We assume that our readers have some knowledge of REXX.

Our goal is not to replace existing documentation, nor to provide a NetRexx reference. We can strongly recommend redbook *Creating Java Applications Using NetRexx*, SG24-2216 to learn about using NetRexx and Java, as well as Mike Cowlishaw′s *The NetRexx Language* for NetRexx reference. The first book gives innumerable examples; in areas such as NetRexx introduction, coding GUI applications, exploiting multitasking.

4.1 REXX′s Position

REXX is often termed as being a *scripting language*. The authors of this book do not agree with this statement. REXX definitely has a broader scope than that.

Surely on VM, the birthplace of REXX, REXX is used by many mission critical applications. So REXX is a true programming language.

On the other hand, we admit that only on VM, REXX can reach everything and everything can reach REXX. On other platforms though, can REXX often only use a subset of the available interfaces, in which case the usefulness of REXX is much reduced.

4.1.1 The REXX Language

REXX is an interpreter, maybe one of the reasons why some people name it "a scripting language." An inherent advantage is that no compile step is required. The cycle is write and test, as opposed to write, compile (= wait) and test. This is fast development, but many errors that a compiler would detect are detected only at runtime.

The REXX language is standard on all platforms, and has a rich function set for string manipulation. Furthermore REXX is a "typeless" language, transparently converting numbers to strings when required. Add to this that variables don′t need to be declared and you understand why REXX is so easy to use.

At the other side, REXX itself provides a limited interface to the outside world: only file reading and writing has been defined. Platforms can (and do) add functions and make other APIs available to REXX, but those extra functions often are not portable.

More recent REXX programs on VM often interface with CMS Pipelines, a very powerful combination, but can then only run on VM (these Pipelines are also available on OS/390, but not widely spread).

© Copyright IBM Corp. 1998

4.1.2 REXX Compilers

On both VM and OS/390 REXX compilers exist; they provide improved performance and protect source code from modification and copying.

4.1.3 Hello World in REXX

Coding a REXX program that says “Hello world” and running it is trivial: create a file with two lines, and type its name to run it. Our “Hello world” program is advanced as it also tells how much 1+1 is.

```
/* This REXX program greets the world */
Say ′Hello world, 1+1=′ 1+1 ′today.′
```

4.2 NetRexx′s Position

NetRexx is a new language, built on top of Java. Its goal is to make writing Java programs easier, and to bring some of REXX′s advantages to the Java world.

NetRexx is not an interpreted language - it must be compiled. The NetRexx compiler translates the NetRexx source into a Java program and then calls the Java compiler for you. As Java is platform independent, NetRexx programs are portable too.

As compiled NetRexx programs become Java classes, NetRexx programs can be called by Java programs also..

Note: NetRexx provides some Java classes as well. Using mainframe technology, one can term these classes the “NetRexx runtime library.”

4.2.1 Hello World in NetRexx

Coding a ″Hello world″ program in NetRexx is a simple as in REXX. Running it is easy too: type “java” followed by the name of the program. But, it must have been compiled.

```
/* This NetRexx program greets the world */
Say ′Hello world, 1+1=′ 1+1 ′today.′
```

4.2.2 Hello World in Java

As a comparison, here is the same ″Hello world″ program written in Java. Beware: Java is case sensitive; except for the string “Hello world.” everything must be coded in the case shown.

```
/* This Java program greets the world */
class HelloWorld {
   public static void main (String args[]) {
     System.out.println(″Hello world, 1+1= ″+ (1+1) + ″ today.″ );
   }
}
```

4.2.3 The NetRexx Language

The base NetRexx language is similar to REXX, and almost all REXX string functions and operators are supported. In addition, the complete Java toolkit is available at your fingertips. This means, for example, that you can now write programs that use such things as multitasking, or GUI interfaces. in a REXX fashion.

Java′s GUI interface, based on the AWT class, is not supported on VM. Even though GUI programs can be written and compiled on VM, they must run on another platform supporting Java′s AWT class, such as OS/2 or the IBM Network Station.

As Java is platform independent, your programs can run on many more platforms.

4.2.4 NetRexx and Compilers

A compiled NetRexx program is a Java program, or a **class file** (to use Java terms). But, using mainframe terms, one can say that NetRexx has a runtime library. In Java terminology: NetRexx provides some class files. The Java virtual machine must be able to find them, else not all NetRexx programs can run.

The NetRexx compiler, first a REXX program, now is a NetRexx program itself. This means you can use NetRexx on any platform with Java support.

Java itself is also a kind of interpretive language. It differs from REXX in that Java does not interpret the source of the program, but a kind of compiled version. The compiled version of the program, class files, are still machine independent. To improve the performance many platforms now provide “Just in Time” compilers that compile a Java class file in machine dependent format just before execution.

4.3 NetRexx Syntax Introduction

A problem while designing NetRexx was that it should appeal to REXX users as well as to Java users. Don′t forget that Java is a pure Object Oriented (OO) environment. REXX and Java are very different; for example, REXX is insensitive to both case and data type, while Java is very sensitive to both.

Combining these two worlds in one language, NetRexx, is surely not easy. And Mike Cowlishaw decided not to make NetRexx 100% upward compatible with REXX. As a result, some shortcomings of classic REXX could be overcome in NetRexx.

We can distinguish a few topics in the REXX and NetRexx language:

Basic syntax Here we classify how to continue a line, how to make a comment line, how to code a string and so on.

Data types REXX is a language without data types (everything is a string). NetRexx allows the use of data types, but gives freedom too.

Case REXX and NetRexx are both case insensitive. The symbols “kris” and “Kris” are the same.

REXX instructions In this topic we discuss the (Net)REXX instructions and a few examples: **do**, **parse**, **if**, ...

Function calls REXX always had a rich set of functions, some examples being **left()**, **translate()**, **word()**, ... NetRexx provides most of them and provides a few new ones. But calling them is a bit different.

Subroutines This is how a programmer can subdivide a program, or re-use software. As OO is all about re-using software, you should not be surprised that here NetRexx differs greatly from REXX.

We go now through these topic one by one. Our goal is not to give a complete list of differences. Our goal merely is that the reader can understand the examples in this book, and to give enough information to start coding. It is fairly safe to assume that items we don't mention are identical in REXX and NetRexx, or of little importance. Furthermore we assume the reader is familiar with REXX.

4.3.1 Basic Syntax Differences

Even though most of the basic syntax of REXX and NetRexx is identical, some differences exist.

Line continuation: In REXX a comma at the end is used to indicate line continuation. In NetRexx a minus sign is used instead:

```
LongLine = 'This is a long line that' -
           'spans two source lines'
catLine  = 'Concatenation is just li' || -
           'ke this'
```

Comments

Comments can be written as in REXX. In addition NetRexx provides an easier way: everything following two minus signs is also a comment.

```
/* This is a comment, just like in REXX */
/* Comments like this can span lines
   as we show here.                     */
-- But this is a comment in NetRexx only
say 'Is it' time  -- tell how late it is
```

The remainder of the basic syntax is identical.

4.3.2 Data Types

REXX is a data type-less language, designed for humans, not for computers. NetRexx follows a similar concept, but it can work with data types as well.

In most classic languages one can basically see two data types only: character strings and numbers.

Object Oriented languages extend the data type principle quite a bit: a *date* data type is an example easily understood, more complex examples are a push button, or even a whole frame. In fact, the data type of an object equals the class of the object. Thanks to this, an OO language can, for example, know that the "add()" method for an object of class "date" is different from the "add()" method for an object of class "frame" and that an object of class "String" has no "add()" method at all. Look at Chapter 5, "AboutFrame, a Reusable Class" on page 55 for an explained class example.

REXX Data Types

To REXX everything is a string. Because of this, there is no need to “declare” variables; REXX defines them when first encountered. A small drawback is that more runtime errors are possible. When issuing: `Say a '+' b 'yields' a+b`

REXX has to convert the strings *a* and *b* to numbers, and that can fail. In a compiled language, the compiler might detect that “a” or “b” is not defined as a numeric data type and produce a compile time error.

NetRexx Data Types

To NetRexx anything that hasn't been predefined becomes an object of class **REXX**, which is similar to Java's **string** class. This is similar to REXX.

When required, NetRexx converts the data type. Again this means that some conversion problems are only detected at run time. Just as in classic REXX, one can do arithmetic with variables that are REXX strings.

In a NetRexx program you cannot only use data types defined by NetRexx, but all data types (or classes) known to Java as well. Native data types are really basic, not implemented as classes. Some of the native data types are:

`boolean, byte, char, int, float`

Declaring Variables

In REXX it is impossible to declare variables before using them. In NetRexx it is possible. A variable that is only declared has a data type, but no content. Here follow some NetRexx definition examples:

```
a = 'test'         -- a becomes an object of type REXX
b = REXX           -- b is declared as a REXX type object
                   -- but, the object does not yet exist
c = string 'test'  -- c is a native Java string
d = string         -- declare a native Java string
e = int 7          -- e is the integer 7
f = Frame('test')  -- f is a Frame with 'test' as title
g = Frame          -- g is a declared Frame
h = myTestClass    -- h is of type myTestClass
```

Not Initialized Variables

As opposed to REXX, not initialized variables are not allowed. Variables must be initialized before being referred to or a compile time error will occur. The two line program

```
hello='Hi there'
say hello to the world
```

displays in REXX “Hi there TO THE WORLD,” but it cannot be compiled in NetRexx. So NetRexx has no need for the `Signal on NoValue` instruction.

Empty Variables

In REXX a variable can be considered “empty” for two reasons:

- The variable referred to is not defined. REXX defines its content then as the uppercase of the variable name. An example of testing this:
  `if Symbol('myVar1')<>'VAR' then ...`
- The variable holds a zero length string. An example of testing this:
  `if myVar2='' then ...`

In NetRexx the situation is different: undefined variables cannot exist (the NetRexx compiler detects them). On the other hand, variables can have a new state: declared only. As for REXX there are two cases:

- The variable referred to is declared only. Using it in a Say yields a program abend: "NullPointerException." Testing for this case can be done as:
  `if myVar1=null then ...`
- The variable holds a zero length string. An example of testing this:
  `if myVar2='' then ...`

Both cases are different; myVar2 is not null.

4.3.3 Case

REXX and NetRexx are both case insensitive. The symbols "kris" and "Kris" are the same.

While comparing strings, the case is preserved. So, when defined as follows, "abc='Test'" and "def='test'," the contents of "abc" and "def" are not identical.

Java class names are case sensitive, so the class file names resulting from a NetRexx compile are also case sensitive. NetRexx will use the case of your class definition.

When using Java classes from NetRexx it is not required to exactly match the case of the Java class name; NetRexx resolves the problem at compile time. First a search is made with the case as coded in the program, and if no match is found a case insensitive search is attempted.

4.3.4 REXX Instructions

Only a few major differences do exist. The new **class** and **method** instructions are handled later.

Grouping Code
The Do Instruction

In REXX the **do** instruction is used for two purposes. Here we discuss the use of Do to group a set of instructions. For example:

```
if something=bad then do
   ...
end
```

This remains the same in NetRexx, except that there are extensions. The most important extensions are the **catch** and **finally** instructions:

```
if something=bad then do
   ...instruction set 1...
  catch <exception>
   ...instruction set 2...
  finally
   ...instruction set 3...
end
```

The normal instructions are `...instruction set 1...`". If the named *exception* occurs, instructions `...instruction set 2...` are executed. An example of an *exception* is "file not found." The instructions `...instruction set 3...` are always executed at the end, both in normal and exceptional conditions.

Looping Through Code
The Do or Loop Instruction

Here we discuss how a set of instructions can be iterated. In REXX we also use the Do instruction. For example:

```
do i=1 by 5 to 99
   ...
end
```

This has been changed: in NetRexx you have to use the **Loop** instruction instead. The *While*, *Until* and *Forever* options exist as well. A NetRexx example:

```
loop i=1 by 5 to 99
   ...instruction set 1...
  catch <exception>
   ...instruction set 2...
  finally
   ...instruction set 3...
end i
```

The Select Instruction

This is compatible, but a Select group can also include the **catch** and **finally** instructions, with the same usage as in a Loop construct.

The Label option

To end the changes for Do, Loop, and Select we must mention the Label option. When nesting **do** blocks in REXX it is sometimes hard to match the corresponding **do** and **end** statements. Only for REXX's **do i=...** variant it is allowed to code **end i**.

NetRexx gives an improvement in this area with the **Label *name*** option. This "Label" option can be used on **Do**, **Loop**, and **Select** to assign a name to the coding block. This name can then be used on **Leave**, **Iterate**, and **End** statements. Two NetRexx examples, side by side:

```
Select Label myTest         | Loop i=1 to 9 Label myLoop
  when xyz=false then do    |   ...
   ...                      |   do label InnerDo
  end                       |      ...
  Otherwise                 |      leave myLoop
   ....                     |   end innerDo
  catch <exception> ....    |   catch <exception> ...
  finally ...               |   finally ...
end myTest                  | end myLoop
```

Note that it is still valid to use the control variable as the name of the group ("i" in the above example). For example:

```
do i=1 to 9 ...  leave i  ... end i
```

The Parse Instruction

The instruction still exists but the defined input sources are limited to the next four:

```
parse term     template    (analyzes "term")
parse ASK      template    (reads an input stream)
parse SOURCE   template
parse VERSION  template
```

```
Where "term" must be a string or result in a string, as:
  parse 'Something is strange' what verb how
  parse what.left(4) begin 2 rest
```

Notice that `parse ARG template`, the format most often used in REXX, no longer exists. The reason is that NetRexx is an OO language in which input parameters for "subroutines" are predefined and pre-parsed (some examples follow). In OO terminology, subroutines are called "methods" as is explained below.

In some examples you will find a `parse ARG ...` instruction. But in these examples the input for the method has been placed in variable "arg," either explicitly or by the "Method" instruction that is added by the NetRexx compiler. The `parse ARG` becomes in fact an ordinary `parse term`.

The **parse upper ...** construct no longer exists. The **Upper** method should be used instead, and (thanks, Mike) there is also a **Lower** method.

Getting Keyboard Input: The REXX instructions to get keyboard input on VM, **parse pull ...** and **parse external ...** can be replaced by either **ask** or **parse ask**. An example:

```
pool = REXX  -- "pool" must be declared before the "Loop"
Loop until pool <> ''
   Say 'Give the name of a swimming pool please'
   parse ask pool  0 test .
   if pool = '' then Say 'I asked you something'
end
test=test.upper()
if test = 'MINERALTHERME' then do
   Say 'Yes indeed the pool' pool 'is great'
   Say 'Do they have a sauna too ?'
   yesno = ask
   if 'YES'.abbrev(yesno.upper(),1) then
      Say 'You must have been there'
end
```

The ADDRESS instruction

Even though many REXX programmers don't know well what the *Address* instruction exactly does, it is used all the time. With the *Address* instruction REXX is told what to do with statements that are neither REXX instructions nor assignments.

The default in a VM EXEC is address CMS which means: "pass those statements to CMS." In an XEDIT macro the default is address XEDIT which instructs REXX to pass those statements to the XEDIT editor. Some examples:

```
address commmand     /* a STATEMENT, so for REXX itself   */
abc='Q DISK'         /* an ASSIGNMENT, for REXX too       */
if xyz<> ''  then    /* 2 STATEMENTs,  for REXX           */
   abc 'A'           /* no STATEMENT, no ASSIGNMENT ...
                        ==> resolved and passed to the ...
                            current "address" = "command" */
```

Well, **NetRexx has no Address instruction**. This means that a NetRexx program cannot execute CMS, CP, or XEDIT commands.

To Remember

This is one of the major differences from REXX: a NetRexx program cannot be used as a command scripting language. Just as C and PL/I, NetRexx is only an application programming language, but a very powerful one.

Through the use of a C or an Assembler program, execution of CMS commands is possible from Java, hence from NetRexx as well.

The good news is that with this redbook a C program is delivered that allows a Java program to execute a CMS Pipeline. And, in CMS Pipelines everything is possible. See Chapter 9, "Java and CMS" on page 93 for more details.

4.3.5 Function Calls

One of the basics any REXX programmer has to know is that a word followed by a parenthesis is a call to a function, and the function will return something. The result can be an empty string, but even that is a result. A REXX example:

```
atext='It happened in 1998'
year=word(atext,4)
 ==> year holds "1998"
```

Functions cannot only be used in assignments. The next example surely is uncommon (and it will end your VM session quickly when executed):

```
subword('The CP LOGOFF command is a fast way out',2,2)
```

This example is important though for another reason.

In general, in REXX it is uncommon to code

```
somefunction(parameters)
```

as it would cause the execution of a host command (see "The ADDRESS instruction"). When the programmer is not interested in the result of the function two alternatives are commonly used:

```
junk = SomeFunction(parameters)
call SomeFunction parameters
```

In NetRexx, however, such code is written all the time, and it will not result in execution of host commands.

NetRexx Function Calls: Just as in REXX, NetRexx comes with many string manipulation functions. The way to use them is different, however. These functions are now called **methods**, and NetRexx uses the same syntax as Java to code a call to a method. Next follows a simple example of extracting the fifth word from a string:

```
==== NetRexx ===================   | ==== REXX ======================
team= 'Bengt Dave Kris Sal Steve'  | team ='Bengt Dave Kris Sal Steve'
ProjectLeader = team.word(5)       | ProjectLeader = word(team,5)
```

REXX people can remember that what used to be the main input to a function is now coded before the function call.

In NetRexx the syntax is different as it is based on another philosophy: a method acts upon an object. In the above example, the object is named “team” and the method is called “word.” The parameters that the word() method needs are specified between parentheses, and define how the method will act upon the object for which it is called.

The example that follows is used to explain more complex calls, one of which is known in REXX as “nested function calls.”

```
==== NetRexx ===================   | ==== REXX ======================
VmRedbook='Written on 1998-02-27'  | VmRedbook='Written on 1998-02-27'
when=VmRedBook.word(3)             | when=word(VmRedBook,3)
-- 'when' holds "1998-02-27"       | /* 'when' holds "1998-02-27" */
                                   |
year=VmRedBook.word(3).left(4)     | year=left(word(VmRedBook,3),4)
-- 'year' holds "1998"             | /* 'year' holds "1998" */
                                   |
Say '*'.left(52,'=')'*'            | Say left('*',52,'=')'*'
Say '*'||year.center(51)'*'        | Say '*'||center(year,51)'*'
Say '*'.left(52,'=')'*'            | Say left('*',52,'=')'*'
```

Both display “1998” in a rectangle:

```
*===================================================*
*                        1998                       *
*===================================================*
```

Let's have a closer look at the `year = VmRedBook.word(3).left(4)` statement. The first interesting part is `VmRedBook.word(3)`: the *word()* method is called for object “VmRedbook.” Word() returns a new string object. This new string object is then acted upon by the *left()* method. Left() returns a newly created string object too, and thanks to `year =`, object *year* refers to it. As we did not declare a data type for *year*, NetRexx assigns it class REXX.

The construct `'*'.left(52,'=')` shows that REXX's string handling methods can act upon a hard-coded string as well (a single * character in our example).

Once one is used to the NetRexx notation, it is easier to read than REXX's notation:

NetRexx simply read from left to right; stop at each dot and interpret; iterate this process. A longer example:

```
In20Century = VmRedBook.word(3).left(2).pos('19')
```

REXX find the innermost function call and interpret, then expand to the left and to the right to find the next function and so on. The same example:

```
In20Century = pos(left(word(VmRedBook,3),2),'19')
```

Nested NetRexx Methods

Worth noting though is that truly nested method calls are encountered in NetRexx too, but with a lesser frequency than in REXX. Here are two short examples; the first shows how the Abbrev method has to be used in NetRexx. The second illustrates the addition of a “Label” object to a frame.

```
1 if 'YES'.abbrev(yesno.upper(),1) then ... -- he said Yes

2 myFrame.add(Label("End time:"))
```

It can be seen that nested method calls are only used when two (or more) objects are used in a single statement.

1 the inner method call reads `yesno.upper()`: the object “yesno” (a REXX string) is translated uppercase. The result becomes a parameter to abbrev() method. Method abbrev() works on object **“**YES**”**.

2 the inner method call reads `Label("End time:")`, which is a call to create a new “Label” object. This new object becomes a parameter to the add() method. The add() method works on object “myFrame.”

4.3.6 Subroutines and User Defined Functions

This is how a programmer can subdivide a program, or re-use software. In REXX, the difference between subroutines and functions is that a function **must** return a result, whereas a subroutine **can** return a result. As this difference is not important here, we will use the more general term *subroutine*.

Without being very far from the truth one could say that methods are subroutines. But, the possibilities of OO methods are much more elaborate than those of subroutines in classic REXX.

Classic REXX allows subroutines to be stored in files other than the main program. But, whereas such external subroutines are indeed callable from any other program, there are quite some drawbacks.

Below follows a list of problems encountered in classic REXX.

- Each time an external routine is called, it is read from disk, which incurs much overhead if the routine is frequently called. Programmers can use VM's EXECLOAD command to load them in storage, but it is not automatic.

  Java will keep methods that are being used in storage, surely as long as the objects using them live.

- When an external routine stops running, all its variables are cleared. So it is very difficult to call an external subroutine several times and have it work on the same task. The external routine will have to save intermediate values, in GLOBALV for example.

In an OO language, methods work on an object, and the class can define which variables live as long as the object lives. So it becomes very easy to divide work in pieces. Here follows a short comparison; refer to Chapter 5, "AboutFrame, a Reusable Class" on page 55 for a complete, working, NetRexx example.

```
==== main REXX pgm ===========   ==== Main NetRexx pgm =========== ====
...                            | ...
rslt=SomeExec('INIT task1')    | task1=SomeClass() -- create object
...                            | ...
rslt=SomeExec('STEP1 task1')   | task1.Step1()     -- perform step1
...                            | ...
rslt=SomeExec('STEP2 task1')   | task1.Step2()     -- perform step2
...                            | ...
rslt=SomeExec('DONE task1')    | task1.Done()      -- tell we're done
...                            | ...
```

With the methods a program can complete the definition of an object in small steps.

- In REXX the variable pool of external subroutines is completely isolated from the main program. There are some tricks by which an external subroutine can fetch (or even update) variables from the calling program, but they are tricks, not architected ways. An example:

```
==== main exec ==============   ==== external subroutine SUBR EXEC ====
address command                |
SomeName='abc'                 |
'EXEC SUBR'                    | address command
                               | 'PIPE VAR SomeName 1|Var SomeVar'
                               | ==> we copied callers "SomeName" into
                               |     or variable "SomeVar"
                               | 'PIPE Literal DEFghi|Var SomeName 1'
                               | exit
Say SomeName                   |
 ==> now displays "DEFghi"     |
```

It works perfectly well, but when such a program needs maintenance, the programmer may forget that some subroutines are accessing the program's variables. A seemingly harmless change to a variable name causes the 'trick' to fail.

In an OO language the programmer of a class can easily decide which variables are public and which private. Public variables can be updated by any program (so mostly not a good idea because changes are not checked). Therefore, one normally defines methods allowing access to variables in a controlled way.

Here's an example of a class with only two variables that can be used from the outside. The first one can be changed freely, the second one only through the methods.

```
=== mySpecialClass NetRexx source ==========
Class mySpecialClass
 Properties public
   myPublicVariable  REXX    -- a public variable
 Properties inheritable
   someVarUnderControl REXX  -- a variable available in the class
 method mySpecialClass()     -- constructor method
   myPublicVariable = '123'  -- set some string in this var
   someVarUnderControl = '*' -- set a default too
 method SetmyPrivVar(hisInput = REXX)
```

```
    if hisInput = ... OK ... then
       someVarUnderControl = hisInput
  method GetmyPrivVar() returns REXX
     return someVarUnderControl

=== Using mySpecialClass from NetRexx ===============
 /* First create an object of type mySpecialClass */
  myThing = mySpecialClass()
 /* Updating public variable is like this: */
  myThing.myPublicVariable  = 'test'
 /* An update using a method is like this: */
  myThing.SetmyPrivrateVar('test2')
 /* Using class variables is straight forward as well: */
  say myThing.myPublicVariable
  Say myThing.GetmyPrivrateVar()
```

Define and Change At Once

Note that the call to a constructor and a method can be coded in single statement.

```
mySpecialClass().SetmyPrivVar('test2')
```

The first part of the statement `mySpecialClass()` is a call to the constructor, so a new object is created. The handle to this newly created object becomes input to the `SetmyPrivVar('test2')` method call. It should be clear that after this statement our program has no handle to the created object. So we can never reference it again, what makes this technique not appropriate all the time.

Parameter Checks

Worth mentioning is also that a basic check of the parameters is made at compile time. With our class definition above, the next statements would give a compile error:

```
myThing.SetmyPrivVar('Calw','city')     -- too many parms
myThing.SetmyPrivVar()                  -- too few parms
Say myThing.GetmyPrivVar('Altensteig') -- too many parms
myThing.Left(4)                         -- undefined method "Left"
```

Because of the mentioned drawbacks, many REXX programmers don't use external subroutines. When starting with NetRexx they should change their habits.

4.3.7 Exit or Return

In an external subroutine, a REXX programmer can code **Exit** or **Return**; both can be used to return to the caller.

In NetRexx, **Exit** means "stop the Java virtual machine" and the whole application stops, not just the method. So NetRexx methods must use **Return**. Note though that when a **Return** is issued in a coding block with a **Finally** instruction, the statements following the Finally instruction are still executed.

4.3.8 Stems - Array Variables - Indexed Strings

The real name for a REXX “Stem” is a “compound variable.” A compound variable is any symbol that includes a period. It is composed of a stem and a tail separated by a period. Some examples;

```
name.1 = 'Koen'             /* stem is NAME ; tail is 1  */
name.2 = 'Karlien'
name.3 = 'Hanne'
son    = name.1
drive  = 'bike'
name.son.drive='velo'       /* ==> NAME.Koen.bike = velo */
mother.    = '--Unknown--'  /* Define a default value */
mother.son = 'Greet'        /* ==> MOTHER.Koen = Greet   */
⋮
drop mother. name.          /* Drop them */
```

From the example one can conclude that a tail does not have to be numeric, and that multiple dimensions are possible.

In NetRexx you can use different kinds of compound variables. But, as the period character is already used in method calls, the period is no longer used to indicate compound variables. Square brackets are used instead.

Using square brackets often causes NLS problems. The expected hex code in EBCDIC for [and] are X'AD' and X'BD'. To verify if your terminal produces the right hexadecimal characters the following pipeline can be issued:

```
pipe literal []|spec 1-* c2x 1|cons
ADBD                    <-- all is OK if ADBD is displayed
Ready;
```

If this poses problems with your terminal, consider to include the next lines in your PROFILE EXEC:

```
'SET INPUT [ AD'
'SET INPUT ] BD'
'SET OUTPUT AD ['
'SET OUTPUT BD ]'
```

Note: To remove these input and output translations, just issue `SET INPUT` and `SET OUTPUT`.

Let's look at NetRexx's first type of compound variables.

Compound Strings

These compare very well with REXX stems, but they can only be used for objects of data type REXX. And the “stem” part must have been assigned a value before one can use it as a “stem.” The same examples as shown above for REXX.

```
name    = ''                  /* Required before using [ ]*/
name[1] = 'Koen'
name[2] = 'Karlien'
name[3] = 'Hanne'
son     = name[1]
drive   = 'bike'
name[son,drive]='velo'
mother     = '--Unknown--'  /* Define a default value, REQUIRED */
mother[son] = 'Greet'
:
mother = null ; name = null   /* Drop them */
```

NetRexx is a bit better than REXX in that between the square brackets you can place an expression.

```
i=1 ; Say 'kids = ' name[i] name[i+1] name[i+2]
/* and even this works as expected */
loop i=1 to 3
   Say 'Mother of' name[i] '=' mother[name[i]]
end i
/* Or, when using a new possibility */
loop index over name
   Say 'Mother of' name[index] '=' mother[name[index]]
end
```

The **Loop Over** construct has no equivalent in REXX. In NetRexx it can be used to get all defined elements of the compound variable, but in an undefined order.

Fixed Size Arrays

This is the second type of compound variable a NetRexx program can use. In this case, tails can only be numbers, and the size must be defined before using the array. But, the data type is not limited to REXX.

REXX programmers beware: **the first element has index number 0**. To obtain the size of the array, the public length variable can be used.

```
abc = int[9]       -- create an array to store 9 integers
xyz = string[13]   -- create an array for 13 strings
square = REXX[5,5] -- a 5x5 array with REXX Strings
say abc.length     -- should display 9
```

Sometimes it is required to declare an array, but the size is only known at execution time. This is possible too.

```
=== myArrayClass NetRexx source ==========
Class myArrayClass
 Properties inheritable
   someArray  REXX[]         -- declare an array of unknown size
 method myArrayClass(i=int)  -- constructor method
   someArray  REXX[i]        -- Now create the array, only then one
                             -- can use it.
 method SetmyArray (i=int , s=REXX)
   if s <> ...bad...
      then someArray[i] = s
```

```
=== Using myArrayClass from NetRexx ===============
 /* First create an object of type myArrayClass */
  myThing = myArrayClass(5) -- ask for 5 entries in "myArray"
 /* Setting an entry using a method */
  myThing.SetmyArray(0,'first city is Freudenstadt')
  myThing.SetmyArray(1,'second city is Freiburg')
  myThing.SetmyArray(4,'last, nice, city is Tuebingen')
```

Java Vectors

If a NetRexx program needs an array of varying, unlimited size, it can create an object of Java's Vector class. Then, however, all access must be done using the methods provided by the Vector class. Here is a simple example or a NetRexx program using a Java vector.

```
nameList=Vector();                      -- create a new vector
nameList.addElement("Kika")             -- add element string objects
nameList.addElement("Sieka")
nameList.addElement("MrRogers")
....
nameList.removeElementAt(1)             -- remove second object,from 0
e =nameList.elements()                  -- get ready to enumerate
Say "List of names:"
Loop While e.hasMoreElements()          -- loop thru the vector
  name = e.nextElement()                --   get next object in vector
  Say "..."  name
  End
```

4.3.9 The main() Method - Input Parameters

When starting the Java virtual machine the name of a class file to execute is passed. The Java virtual machine passes control to the method named **main()**. Therefore, any Java program that must be started from the console needs such a method; programs without a "main" method can only be used from another Java class.

It is possible to pass arguments to the started program, where "main" can analyze them.

Parameters for main(), NetRexx Style

When in a NetRexx program no Class statement exists, the NetRexx compiler inserts a Class statement and a "main" method. It also places all arguments in variable "arg." As a result of these conventions, the following is a complete, valid NetRexx program:

```
parse arg word1 word2 word3
Say 'The first word you gave is =' word1
Say 'The second word you gave is=' word2
Say 'The third word you gave is =' word3
```

However, explicitly coding a Class statement is required when your program needs to define some class Properties.

Parameters for main(), Java Style: When coding the “main()” method yourself, Java requires that it read: `Method main(args=String[])`. **Beware**, “main” is case sensitive, even in a NetRexx program: because Java is calling us, the search is not case insensitive.

In a first attempt you could change the above program a tiny bit:

```
Class MyPgm
 Method main(arg=String[]) public static
  parse arg word1 word2 rest
  Say 'The first word you gave is =' word1
  Say 'The second word you gave is=' word2
  Say 'The remainder is          =' rest
```

This however doesn't work: the input is an array of Java String objects.

The `parse arg` instruction compiles OK as variable “arg” exists. But, “arg” contains the handle to an array, and the parse might be analyzing something like `[Ljava.lang.String;@a4487` instead of the program's input parameters.

A second attempt is a bit better:

```
Class MyPgm
 Method main(arg=String[]) public static
  Say 'The first word you gave is =' arg[0]
  Say 'The second word you gave is=' arg[1]
  Say 'The third word you gave is =' arg[2]
```

But a runtime exception is thrown when, for example, the startup parameter contains only two words, and the program attempts to address a third one: `arg[2]`.

Here's a better example:

```
Class MyPgm
 Method main(args=String[]) public static
 -- Place all arguments in a REXX style string:
 arg=''
 loop i=1 to args.length
    arg=arg args[i-1]
 end
 -- Parse the REXX style string:
 parse arg word1 word2 word3
 Say 'The first word you gave is =' word1
 Say 'The second word you gave is=' word2
 Say 'The remainder is          =' rest
```

4.3.10 Comparing NetRexx to Object Oriented REXX

Even though Object Oriented REXX (OO-REXX for short) is not available on VM, it may be worthwhile to highlight a major difference between OO-Rexx and NetRexx.

OO-REXX is indeed another flavor of REXX, and it is an OO language also. The good thing about OO-REXX is that it is upward compatible with classic REXX.

But with Java one gets a rich set of classes, all of which can be used by NetRexx. OO-Rexx cannot use the Java classes.

Chapter 5. AboutFrame, a Reusable Class

In this chapter we use a NetRexx class file, written during the project, to explain more basics of the NetRexx language, Java classes and OO principles.

Even though VM/ESA does not natively support Java's GUI, the AWT class, the example used in this chapter uses a GUI. We specifically chose it for its illustrative value as an example: a GUI program uses quite a few classes, and many of the objects are visualized, making the discussion less theoretical.

5.1 The AboutFrame Picture

First of all a picture of a running instance of the AboutFrame Class is shown. Figure 9 shows AboutFrame as called by the GuiMon sample application.

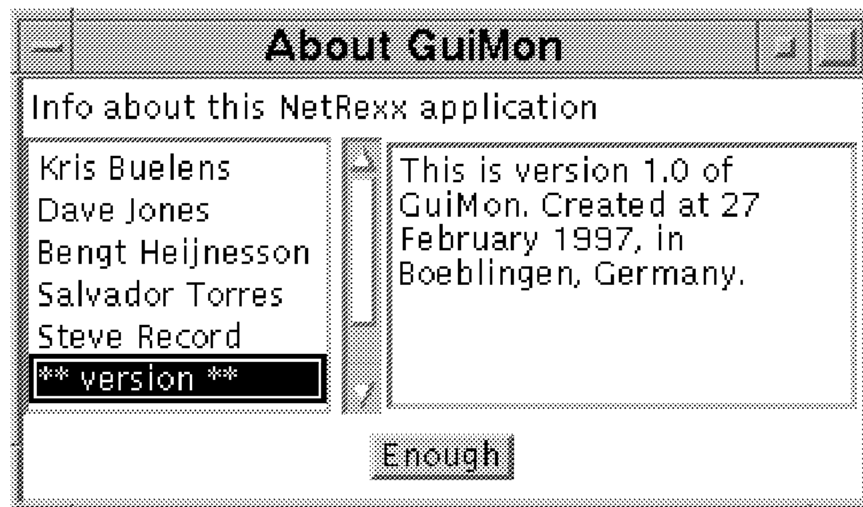


Figure 9. AboutFrame, Displays Information. Version information is currently selected.

5.2 What is AboutFrame?

Many GUI applications have an "about" panel, used to display various information, such as the program's version and the authors.

Eager to become famous, we wanted to provide a similar frame. But as our goal must be broader than just fame, we decided to write the code required for such a frame in a reusable way. As this code is fairly simple, it is used to explain some OO principles. It also illustrates how reusable NetRexx code can be created and used.

First of all the requirements for AboutFrame from an end-user perspective are given. Then programming aspects are discussed.

© Copyright IBM Corp. 1998

5.3 AboutFrame: User Interface

The AboutFrame class provides an advanced about frame, in which the end-user cannot only see author names, but also some extra information about the authors.

At the right side of the frame, a list box lists all authors. When the end-user selects an author a first piece of information related to the author is shown at the right side of the frame. When the end-user double clicks on an author, the second piece of author information is shown.

Note: Program version information is also listed as if it were an author.

5.4 AboutFrame: Program Interface

For an application to use AboutFrame it must be able to provide a list of authors, and for each of them one or two strings of information. In addition, some tailorable title information is welcome.

5.4.1 Approach with Classic Languages

When deciding that the AboutFrame routine must be usable from other programs, a problem is found immediately: how do we pass many different parameters, when the number of parameters is not even fixed. Obviously one could pass all this information in one long string and define some separation characters. This though is not practical, nor a structured way of working.

Let′s concentrate on other solutions a REXX programmer may attempt.

Imbedding the AboutFrame Code as a Subroutine

This is a possibility, all variables of the main program can be exposed in the subroutine. So a REXX "stem" variable could be used. But, when the AboutFrame program is changed, all other programs imbedding it must be changed too. Problems arrive when AboutFrame and the main program use the same variable names. One can use REXX′s `Procedure Expose` instruction to isolate most of the subroutine′s variables from those of the main program.

Practice does however reveal that this is not always easy. Surely when subroutines start calling each other the list of exposed variables can become lengthy. So most REXX programmer no longer even attempt to isolate variables.

Using CMS Pipelines to Obtain the Variables

This is indeed a -relatively new- possibility. But it is cheating a bit: for readers of the main program it may be hard to guess that some of the program′s variables are also used by another program. See 4.3.6, "Subroutines and User Defined Functions" on page 47 for a discussion.

5.4.2 An OO Solution

With an OO language, this problem can be solved easily.

We define a class for the AboutFrame. The class file can be compared to an external subroutine in classic REXX.

- It holds definitions for variables which are used to build an AboutFrame object. The general name in OO terminology for such variables is “object attributes.” In NetRexx they are termed **properties**.

  The properties that are most easily understood are those holding variable information that a program using the class must pass.

- The class definition also includes coding groups. These coding groups are named **methods**. Each method can be called separately. One can compare a method of a class with a function in classic REXX.

  Methods are used to create an object of the class as well as to extract or update the properties of the object(s). Often a method is very simple, containing only a few lines of code.

  Depending on the design of the class, a program using it can build an object in bits and pieces. With each call to a method, the definition of the object becomes more complete.

For our AboutFrame, we can easily distinguish a few properties, just by looking at Figure 9 on page 55, top down and left to right:

- The title of the frame
- The information message
- The list of authors
- The information about each author

Before discussing the actual implementation of the AboutFrame class, some more theory about classes and methods is appropriate. Most of the following theory is illustrated when discussing AboutFrame in 5.6, “AboutFrame: the Class Definition” on page 60.

5.5 Classes and Methods

In this section, some more OO principles are explained, with NetRexx as the language. As we want a broad public we don′t want to be 100% exact or complete. This would frighten readers without OO skills.

When in the text below we write “our users,” “the users,” or “the class users” we refer to the application using the class, not the end-user using an AboutFrame.

5.5.1 Class - What is it?

A class can be seen as a plan to build an object, it is not an object yet. It defines variables required to build and manage the object, as well as methods that the class user calls to interact with the object of the class.

In other words: a class just defines how an object of that kind (= class) can be built and how it can be handled (= what methods do exist). The code itself of the methods is also included in the class file.

Class Instance

When an object of the class is created two terms are used: *the object is an instance of the class* and *the class has been instanciated*. To give a practical example: it is not because we -the team that wrote the redbook- provide the

AboutFrame class, that somewhere in the world an object of class AboutFrame is actually active at this time. The class exists, the objects may or may not.

Class Types

Different types of classes can exist. Multiple classes can be defined in the same source file. User defined classes can extend already defined classes.***:*** Some important class types are:

Public classes can be used by any program.
Private classes can only be called by code in the same source file.

5.5.2 Methods - What are they?

Methods define ways to create an object of the class ("instanciate the class") and to work with the created object. A class mostly needs more than just one method.

One of the reasons that an OO language solves AboutFrame′s parameter problem is that many methods can be defined: one to pass each set of information. Each of the methods can work on the same object, until it is built completely.

Besides the methods that must be available to our users, the class itself may need methods for its own use. Basically we can distinguish two kinds of methods:

Public methods can be used by any program using the class.
Private methods can only be called by code of the class itself.
Inheritable methods can be called by code of the same class, or subclasses.
Static methods work on class level, and not on an object.

There is more to be said about methods, such as what a constructor method is.

5.5.2.1 Constructor Method

This method is always required: a class is a definition plan for an object, not an object yet. When a program wants to use our services, it first must call a constructor method of the class.

The name of the constructor method is the same as the class name. This method may or may not need parameters.

The code in the constructor method actually creates an object and may define some variables for it. A constructor returns the **handle** to the created object. A handle is a kind of pointer to the object.

Handle

When the constructor method ends, the class has *instanciated* an object of the class. Very often a class allows many objects of the class to be instanciated. So each time another class method is called, the caller has to pass the "handle" of the instance that must be addressed. If the caller loses the handle, the object can no longer be reached.

In the AboutFrame example, the constructor creates a standard Java Frame object and populates it with the list box and alike. We also create empty arrays to hold all authors, amongst other things.

5.5.2.2 Other Methods

These methods often are the only way to pass data between the class user and the object. Such methods compare very well with REXX functions, but as opposed to REXX functions, some methods return data, others don't.

In our example we provide at least three methods (and the code implementing the method): define an author: setAuthor(...), define the frame's top information line: SetApplText(...) and a method to start displaying the frame: ShowAbout().

5.5.2.3 Method Arguments - Signatures

When a method is called, the caller can (or must) specify arguments. The number and data type of these arguments must match a definition of the method. For example, the Substr() method of class REXX **might** have been defined in three formats:

Substr(nr) with one number as input.
Example: `Say name.Substr(5)`

Substr(nr,nr) with two numbers as input.
Example: `Say name.Substr(5,8)`

Substr(nr,nr,char) with two numberS and a padding character.
Example: `Say name.Substr(5,8,'-')`

This means that a call to Substr() without any parameter would fail at compile time, just as a call with four parameters or a call with a string instead of a number as first parameter. The example above illustrates that a method with a given name can be defined with multiple ***signatures***. The definition could basically appear so:

```
  Class REXX ...
   method Substr(start=int)
:
   method Substr(start=int,lng=int)
:
   method Substr(start=int,lng=int,pad=char)
:
```

Note though that NetRexx wants to be data type-less, so the Substr() method has been defined with one, two or three REXX strings as input. Only at runtime does NetRexx verify that the first two can be converted to a number and that the third is a single character.

5.5.2.4 Handle - Parameters - The this Variable

Except for the call to a constructor method, a method must always be called for an object ("for an instance of the class"). This is done by writing the handle before the name of the method; parameters are specified between brackets after the method.

```
1 name=REXX('Greet Hermans')  -- create the "name" object
2 Say name.Substr(1,5)        -- call method Substr()
```

1 We explicitly call the constructor to build a brand new REXX string object, passing one parameter. The shorter form `name='Greet Hermans'` would produce the same result, but that a constructor is called is less clear. Anyway, the result is that "name" is a handle to the REXX string object.

2 When we code `name.Substr(...)` the Substr() method knows which object it must act upon. The method does not get the handle to the object as a parameter; when required it can refer to it by using the ***this*** variable.

5.5.3 Variables in the Class

Just as there are different kinds of methods, a class can have different kinds of variables. So all variables do not have the same scope, cannot be "seen" everywhere, do not have the same lifetime. Three basic types can be distinguished.

Local variables

By default, all variables used by a method are "local," which has a few consequences:

- When the method ends all local variables are deleted.
- Other methods cannot reach them, not even methods of the same class.
- When the same method runs concurrently (for two different objects), each instance of the method has its own set.

Method arguments

All arguments passed to a method are local variables too.

Properties

Alongside the class definition instruction, one can define variables that are not local, they all survive the end of the method using them. Such variables are the "Properties" of the class. It can be said that Properties are global variables, but different types exist. Below is a list of a few commonly encountered types.

Constant Inheritable These variables belong to the class, live as long as the class lives, cannot be changed once the class is activated, and are visible to all methods of the (sub)class.

Static Inheritable These variables are similar to those above, except that they can be changed by class methods. Such variables could be used, for example, to keep track of how many objects of the given class have been created.

Inheritable Such variables go with an instance of the class (or with an object). They are visible to the (sub)class methods.

For example: it is possible that two programs would create an AboutFrame object (one program using two about frames is similar). Then there are two instances of the AboutFrame class. Each frame should have its own set of authors.

We can now look at the actual code of the AboutFrame class.

5.6 AboutFrame: the Class Definition

Even though the AboutFrame is relatively small, it must be analyzed in smaller pieces. Therefore Figure 10 on page 61 only prints the main parts. Many lines have been removed. The complete source for the AboutFrame may be found in the sample program set for this book.

5.6.1 AboutFrame: Overview of the Program

Before going into too much detail, here is an overview of the NetRexx program source. Remember that in Figure 10 quite some source lines have not been printed. What is printed should allow the reader to see that we define:

- Variables required to build and manage the frame
- Methods by which the calling program passes the required author information
- Methods to act on end-user actions in the frame. In our case, we need methods to react to end-user requests (such as the selection of an author).

```
1 --NRC-XEDIT-DEFAULTS NoCrossRef
  --NRC-XEDIT-OPTIONS  NoRun
 ------------------------------------------------------------------
2 class AboutFrame extends Frame

3  Properties inheritable
    LstWho  = List(5)           -- define a List Box
    authors = int               -- declare number of authors
    authDesc = String[]         -- declare this array
    authDescL= String[]         -- declare this array
4 Method AboutFrame(MaxAuthors=int)
  -- As our class is a frame extension, we should not call Frame()
  -- else we'd create yet another frame
  --    win = Frame("About")
  -- But, we can call the our parent class to set a frame title.
   nbrFrames=nbrFrames+1
   super.SetTitle("About")  -- define default title
   win=this

   AuthDesc = String[MaxAuthors]          -- create this array
   AuthDescL= String[MaxAuthors]          -- create this array
   authors=-1                             -- first arrays item = 0

   TxtAppl=Label('The application has been written by')
   win.add("North",TxtAppl)
   win.add("West",LstWho)                 -- add List Box to the frame

   setSize(300,200)                 -- define size of window.
5
  Method SetTitle(t=String)  -- Define title of the window
   super.setTitle(t)         -- Must be preceeded by "super" else we
                             -- call ourselves

  Method SetApplText(t=String) -- Define the title of the window
   TxtAppl.setText(t)
```

Figure 10 (Part 1 of 2). AboutFrame, Program Overview

```
   Method SetAuthor(Author=String,Descript=String)
    authors=authors+1
    LstWho.add(Author)
    authDesc[authors]=Descript
   Method SetAuthor(Author=String,Descript=String,DescLong=String)
    authors=authors+1
    LstWho.add(Author)
    authDesc[authors]=Descript
    authDescL[authors]=DescLong
   Method SelectAuthor(ix=int)
    if ix<=authors then LstWho.select(ix)

   Method ShowAbout()
    -- As "we" the object are in fact a frame, no need to code
    -- this.setVisible(...)  or win.setVisible(...)
    setVisible(1)
 --------- This class handles Action Events with objects in the frame
6 class AboutActionClass implements ActionListener,ItemListener
    Properties inheritable
      frm         = AboutFrame -- frm is an object of class AboutFrame
      myEventName = String -- a string is passed and available in the class
   -- Constructor
   method AboutActionClass(x = AboutFrame, anEvent = String)
      frm         = x
      myEventName = String anEvent
   method itemStateChanged(e = ItemEvent)
    ix=frm.LstWho.getSelectedIndex()  -- Get the selected line -if any
    Say 'Select event in Listbox, item:' ix

    if ix >= 0 then frm.TxtWho.setText(frm.authDesc[ix])
              else frm.TxtWho.setText(' ')
```

Figure 10 (Part 2 of 2). AboutFrame, Program Overview

Below are a few words about the areas referenced in the figure. Remember that with this figure we only want to give an overview of the program. More details follow in later figures.

1 The NRC XEDIT macro, explained in 3.4.2, "NRC XEDIT - NetRexx Compile" on page 30, that we use to compile NetRexx programs reads these two comment lines to find runtime options. In this case we ask not to produce a variable cross reference and not to run the program after a successful compilation. It is indeed impossible to run AboutFrame on VM as it uses Java's AWT class, which is not supported on VM.

2 **Class** is the first real NetRexx statement. We define the class that the NetRexx compiler should build. If you don't code a Class statement, the NetRexx compiler will insert one. Coding a Class statement is required if it needs extra options or Properties are coded.

3 Following the Class statement, **Properties** for the class are defined. Remember that with properties we define variables that are not local to a single method. All properties must be defined before the first method.

4 This is a constructor method as it has the same name as the class itself. Notice that following this **Method** instruction some real coding follows: a frame is built and some property variables get a value.

5 Here other methods are defined. All methods of our class follow, with their NetRexx code.

6 Here we start a new class. This illustrates that in one source file more than a single class can be defined. The class defined here has its own methods, they are used to react to events on the frame, such as the end-user selecting an author in the list box.

Having seen the structure of a NetRexx source, sections of the program are analyzed one by one below. We no longer remove some code from the printout.

5.6.2 AboutFrame Section One: The Class Itself

In Figure 11 we concentrate on the class definition and its properties.

```
1 class AboutFrame extends Frame
2  Properties inheritable static -- some vars for the whole class
       NbrFrames = 0
3  Properties inheritable
       TxtWho  = TextArea(' ',40,90,TextArea.SCROLLBARS_NONE)
       LstWho  = List(5)           -- define a List Box
       TxtAppl = Label             -- declare read-only text
       PbtCncl = Button('Enough')  -- define a push button
       authors = int               -- declare number of authors
       authDesc = String[]         -- declare this array
       authDescL= String[]         -- declare this array
```

Figure 11. AboutFrame, Class Definition and Properties

1 “AboutFrame” is defined as a class. The **extends Frame** parameter indicates that AboutFrame inherits from Java's “Frame” class. In other words: an AboutFrame is a Frame, but a special one; one with special properties.[2] As AboutFrame has “Frame” as its super Class all Methods of the Frame class can be used on an AboutFrame object.

2 The Property “NbrFrames” is defined as “Static,” meaning it exists only once for the whole Class. But as it is inheritable, all Methods can access it. We use it to count the number of AboutFrames created this far. As we code `NbrFrames = 0`, it will be initialized to 0 as soon as the AboutFrame Class is activated.

3 Here we define the other inheritable Properties. All variables that must survive the end of a Method must be defined here (the “authDesc” strings array is an example). Another reason to define a variable as a Property is when it must be used by more than one Method (the handle to the list box “LstWho” is an example).

Before going to the next section of AboutFrame, notice that some Properties are declared only. For example the “authDesc” array can only be declared as the size of the array is not fixed. It is created when an AboutFrame object is created.

[2] The above sentence illustrates very well what a Property really is. In this case, the word “property” can be interpreted in its normal English meaning, or taken as NetRexx's term.

5.6.3 AboutFrame Section Two: The Constructor Method

In Figure 12 the constructor method for the AboutFrame class is discussed.

```
1  Method AboutFrame(MaxAuthors=int)
      nbrFrames=nbrFrames+1 method AboutFrame(MaxAuthors=int)
      Say 'Constructing the frame, max authors=' MaxAuthors -
          'this is AboutFrame nbr' nbrFrames
   -- As our class is a frame extension, we should not call Frame()
   -- else we'd create yet another frame
   --  win = Frame("About")                  --  should not be coded
2     win=this                                -- "win" is nicer than "this"
   -- We can call the parent class to set a frame title.
3     super.SetTitle("About")  -- define default frame title

4     AuthDesc = String[MaxAuthors]          -- create this array
      AuthDescL= String[MaxAuthors]          -- create this array
      authors=-1                             -- no authors defined yet
      -- To close the window from the system menu work with WindowListener
5     anObject = AboutFrameController()       -- Create this object

   -- Because "we" are a frame, no need to write "win.add"
   -- but it may be clearer for some readers.
      win.addWindowListener(anObject)
   -- To be able to react to end-user frame events
      pbtCncl.addActionListener(AboutActionClass(this,'pbtCncl'))
      lstWho.addActionListener(AboutActionClass(this,'lstWho'))
      lstWho.addItemListener(AboutActionClass(this,'lstWho'))
6  -- add the visible objects to the frame
      TxtAppl=Label('The application has been written by')

      TxtWho.setEditable(0)         -- make this area read only
      win.add("North",TxtAppl)      -- add these objects to the frame
      win.add("West",LstWho)
      win.add("Center",TxtWho)
      p=Panel()            -- Host the button in a "panel" to keep it small
      win.add("South",p)   -- add the "panel" to the frame
      p.add(PbtCncl)       -- place the button in the "panel"
   -- use some colors
      hYell = color(255,255,128)             -- define a color object
      TxtAppl.setBackground(color.white)     -- color of application text
      setBackground(color.white)             -- color of our frame
      pbtCncl.setBackground(color.lightGray) -- color of our button
      LstWho.setBackground(hYell)            -- color of our LISTBOX
      TxtWho.setBackground(hYell)            -- color of our text area
```

Figure 12 (Part 1 of 2). AboutFrame, Constructor Method

```
7 ----- calculate a good place for our frame ---------------
     setSize(300,200)                     -- define size of window.
     offset= (NbrFrames-1) *10            -- don't place all at same place

     d = getToolkit().getScreenSize()     -- get size of the screen
     s = getSize()                        -- get size of our frame
     SetLocation( (d.width - s.width)  %6 + offset, -
                  (d.height - s.height)%6 + offset   )
```

Figure 12 (Part 2 of 2). AboutFrame, Constructor Method

1 The constructor method is defined. AboutFrame provides only one constructor: an integer is expected as parameter. This integer is the number of authors the AboutFrame must be able to handle. This integer is automatically placed in local variable ″MaxAuthors″ when the method starts. Notice again that unlike REXX, NetRexx doesn′t use “parse arg” to obtain the method′s arguments.

2 As “Frame” is the parent class of AboutFrame, any instance of AboutFrame is a Frame already and the variable “this” points to it. If we would explicitly code `win=Frame(...)`, a second frame object would be created, and “win” would point to it. This can lead to problems, refer to 5.6.6, “Avoiding Empty Frames” on page 73 for more details.

3 With this statement NetRexx is explicitly requested to call the SetTitle method of class “Frame,” our super class. If the statement had been “SetTitle(...)” our own method “SetTitle” method would have been called (see Figure 13 on page 66 item **1**).

4 As mentioned above, the “AuthDesc” array has been declared as a property, only now it will actually be created, with the argument passed by our user as its size. As “AuthDesc” is an “inheritable property,” each instance of AboutFrame gets its own copy of this array, all copies can have a different size and different contents.

5 When programs needs to react to end-user actions on a frame, the program needs to provide special methods in other frame related classes. More information is provided in Figure 14 on page 67.

6 In this section, the layout of the frame is defined. Without going into details, the default frame layout has five places, each is named to an orientation: North, East, Center, West, and South. Each place can get one object; when more objects are required, a Panel object can be used and subdivided again. A good section about window layout strategies can be found in redbook *Creating Java Applications Using NetRexx*, SG24-2216, chapter 7. Another, more complex, layout example can be found in the GUIMON program which is discussed in Chapter 10, “The GUIMON Sample Program” on page 101.

7 The last section discussed here is the sizing of the AboutFrame and its placement on the screen. We make the placement depend on the size of the terminal being used. Without extra precautions, all AboutFrames would be placed at exactly the same place, and the last one would hide all others. Therefore, an extra offset is calculated, and multiple about frames are shown as a cascade.

Sample Call Format

A NetRexx programmer can create an AboutFrame by coding:

```
myAbout = AboutFrame(7) -- Create an AboutFrame, 7 authors
```

5.6.4 AboutFrame Section Three: Other Methods

In Figure 13 the non-constructor methods of AboutFrame are printed.

```
1 Method SetTitle(t=String)  -- Define title of the window
    super.setTitle(t)         -- Must be preceeded by "super" else we
                              -- call ourselves

   Method SetApplText(t=String) -- Define the title of the window
    TxtAppl.setText(t)

2 Method SetAuthor(Author=String,Descript=String)
    authors=authors+1           -- add an element to the authors list box
    LstWho.add(Author)
    authDesc[authors]=Descript -- add his description to array

3 Method SetAuthor(Author=String,Descript=String,DescLong=String)
    authors=authors+1
    LstWho.add(Author)
    authDesc[authors]=Descript
    authDescL[authors]=DescLong
   Method SelectAuthor(ix=int)
    if ix<=authors then LstWho.select(ix)

4 Method ShowAbout()
    -- As "we" the object are in fact a frame, no need to code
    -- this.setVisible(...)  or win.setVisible(...)
    setVisible(1)
```

Figure 13. AboutFrame, Other Methods

1 The first method deserving some attention is *SetTitle*, which allows our user to define the title for the AboutFrame. The only parameter the method gets is placed in local variable *t*; its data type must be a String. The method is very simple: it calls the SetTitle method in AboutFrame's super class Frame. So the method is completely useless: if we wouldn't have provided this method, the programmer could still code `myAbout.SetTitle('About xyz')`. NetRexx would have found that Frame, the super class of AboutFrame has a SetTitle method and call it. We opted to code the method as it gives the opportunity to explain that one can override methods.

2 and **3** illustrate a method, SetAuthor, with two signatures: the first one accepts two strings, the second one is called when the user codes three strings as parameters. The code for **3** could have been shortened by coding:

```
Method SetAuthor(Author=String,Descript=String,DescLong=String)
 SetAuthors(Author,Descript)
 authDescL[authors]=DescLong
```

4 When our user has completed the building process of his AboutFrame he should make the frame visible by calling the ShowAbout method.

Sample Call Format

Some examples of calls to AboutFrame′s methods are:

```
myAbout.SetTitle(′About xyz′) /* Set frame title  */
myAbout.SetApplText(′Information about ...′)
myAbout.SetAuthor(′Mr x′,′is the author′)
myAbout.SetAuthor(′Mrs G.′,′is co-author.′,′Born in Belgium′)
myAbout.ShowAbout()
```

5.6.5 AboutFrame Section Four: Event Classes

First closing a frame is covered, then other events are handled.

5.6.5.1 Principles of Window Closing

For completeness we repeat a section from Figure 12 on page 64 in Figure 14.

```
  Class AboutFrame  extends Frame
  ⋮
   Method AboutFrame(MaxAuthors=int)
  ⋮
1  -- To close the window from the system menu work with WindowListener
   anObject = AboutFrameController()       -- Create this object

   -- Because ″we″ are a frame, no need to write ″win.add...″
   -- but it may be clearer for some readers.
2       win.addWindowListener(anObject)
&invellipsis.
3 class AboutFrameController extends WindowAdapter
     method windowClosing(e = WindowEvent)
       Say ′Closed by system menu′
       exit
```

Figure 14. AboutFrame, Event Classes

Statements **1** and **2** allow the user to close the AboutFrame window by double clicking in the upper left corner of the frame.

Java will call Method WindowClosing of Class WindowAdapter, or a class extending it, when the user wants to close the window. Java′s default Method ignores the request, so we have to code our own Class and Method, overriding Java′s (**3**).

1 As our action routines are classes, they don′t exist as long as a constructor method isn′t called. Here we explicitly call the constructor of the AboutFrameController class, to create an instance of the class and get the handle to it. The handle to the instance is stored in “anObject.”

2 We add the created instance of AboutFrameController as “windowListener” to the frame object. Statements **1** and **2** could have been combined in a single, even more obscure, statement:
`addWindowListener(AboutFrameController())`

3 The class AboutFrameController and its methods are defined here. The name of the class can be selected at will; the names of the methods are predefined by Java. A simple interpretation of the Class keyword *extends* in combination with

Method *windowClosing* could be: “we don’t like the actions of the default method, hence we create our own, overriding the default.”

5.6.5.2 Handling Other Events

For completeness we repeat a section from Figure 12 on page 64 in Figure 15.

```
  Class AboutFrame  extends Frame
  :
   Method AboutFrame(MaxAuthors=int)
  :
    -- Make handing other events possible
 1    someObject=AboutActionClass(this,'pbtCncl')
 2    pbtCncl.addActionListener(someObject)
 3         lstWho.addActionListener(AboutActionClass(this,'lstWho'))
       lstWho.addItemListener(AboutActionClass(this,'lstWho'))
  :
 4 class AboutActionClass implements ActionListener,ItemListener
     Properties inheritable
       frm         = AboutFrame -- frm is an object of class AboutFrame
       myEventName = String -- a string is passed and available in the class
    -- Constructor
 5  method AboutActionClass(x = AboutFrame, anEvent = String)
       frm          = x
       myEventName  = String anEvent
 6  method actionPerformed(e = ActionEvent)
       Say 'Event happened for:' myEventName
      Select
       when myEventName='pbtCncl' then do
        Say 'Closed by Cancel button'
        frm.dispose()
        return
       end
       when myEventName='lstWho' then do -- double click in List Box
    -- Beware: testing if ..[] = '' is dangerous, it may yield a
    --        null pointer exception.  So test for "null"
          ix=frm.LstWho.getSelectedIndex()  -- Get the selected line
          t= ''
          if frm.authDescL[ix] <> null then
             if frm.authDescL[ix] <> '' then
                t=frm.authDescL[ix]
          if t<>'' then do
             frm.TxtWho.setForeground(color.black)
             frm.TxtWho.setText(t)
          end
          else do
             frm.TxtWho.setForeground(color.red)
             frm.TxtWho.setText('More information about' -
                                 frm.LstWho.getSelectedItem()-
                                'is not available')
          end
       end
```

Figure 15 (Part 1 of 2). AboutFrame, Event Classes

```
      Otherwise
       Say 'Problem:' myEventName 'is unknown'
     end
7 method itemStateChanged(e = ItemEvent)
     Select
      when myEventName='lstWho' then do
       ix=frm.LstWho.getSelectedIndex()  -- Get the selected line -if any
       Say 'Select event in Listbox, item:' ix

       frm.TxtWho.setForeground(color.black)
       if ix >= 0 then frm.TxtWho.setText(frm.authDesc[ix])
                  else frm.TxtWho.setText(' ')
      end
      Otherwise
       Say 'Problem:' myEventName 'is unknown'
     end
```

Figure 15 (Part 2 of 2). AboutFrame, Event Classes

When a program needs to react to events in the frame, the program needs to "enhance" other, predefined Java classes. The keyword **implements** (see **4**) is required on the class here.

Coding handlers for other events is similar to the code to close frames: Classes and Methods have to be defined and instanciated. The instanciated classes have to be added as "listeners" to the objects.

1 At this line the constructor of AboutActionClass is called, with two parameters: `this`, which is the handle to the AboutFrame; and a string. The constructor method **5** saves these parameters in two inheritable properties: "frm" and "myEventName." This means that each instance of AboutActionClass has its own copy of it. Note that it is very important that the handle to the frame ("this") is passed to the methods, otherwise the methods cannot know which instance of AboutFrame needs to be addressed (remember than more than one AboutFrame can be active at the same time).

2 Here, the AboutActionClass object is added as a "listener" to the push button named `pbtCncl`. Java's GUI support can then call the ActionPerformed Method of class AboutActionClass when the end-user selects the push button.`pbtCncl`.

3 For the list box, similar coding is created, but, both creating an AboutActionClass object and adding it as a "listener" are done in a single statement.

4 Here our Class to handle events of objects in the frame is defined. The Class keyword *implements* has another meaning than *extends*. It means that the classes ActionListener and ItemLister only define the methods and their parameters, but that our class has to provide the actual program logic.

5 Is the constructor of AboutFrame's class to handle end-user events. We save the parameters in inheritable properties.

6 In AboutFrame, method ActionPerformed can be called for two cases:

- The user selected the "Enough" push button (named "pbtCncl").
- The end-user double clicked on a line in the list box.

The code in the method "knows" what event it was by a string that has been saved at construction time of the AboutActionClass objects.

7 The second method, ItemSelected, is called when the end-user selects a line in the list box.

Note: The action handling code could have been written differently. Our REXX experience means that we often use `if` or `select` statements. Programmers with more OO experience might code more Classes as illustrated in Figure 16.

```
  :
   -- To close the window from the system menu work with WindowListener
   win.addWindowListener(AboutFrameController())
   -- To be able to react to other events
   pbtCncl.addActionListener(AboutClassPbtAction(this,'pbtCncl'))
   lstWho.addActionListener(AboutClassLstAction(this,'lstWho'))
   lstWho.addItemListener(AboutClassLstItem(this,'lstWho'))
  :
--------- This class handles events on the frame itself  ----------
class AboutFrameController extends WindowAdapter
  method windowClosing(e = WindowEvent)
    exit

--------- This class handles Action Events for all Push Buttons
class AboutClassPbtAction implements ActionListener
 Properties inheritable
   frm         = AboutFrame -- frm is an object of class AboutFrame
   myEventName  = String  --a string is passed and available in the class
 method AboutClassPbtAction(x = AboutFrame, anEvent = String)
   frm         = x
   myEventName  = String anEvent
 method actionPerformed(e = ActionEvent)
  -- as we only have one button no need for "if myEventName='pbtCncl' .."
    frm.dispose()
```

Figure 16 (Part 1 of 3). AboutFrame, Event Handling with More Classes

```
--------- This class handles Action Events for all List Boxes
class AboutClassLstAction implements ActionListener
 Properties inheritable
   frm         = AboutFrame -- frm is an object of class AboutFrame
   myEventName  = String  --a string is passed and available in the class
-- Constructor
 method AboutClassLstAction(x = AboutFrame, anEvent = String)
   frm         = x
   myEventName  = String anEvent

 method actionPerformed(e = ActionEvent)
   -- Only one LstBox, so no need for an "if myEventName='lstWho' .."
   ix=frm.LstWho.getSelectedIndex()  -- Get the selected line
   t= ''
   if frm.authDescL[ix] <> null then
      if frm.authDescL[ix] <> ''  then t=frm.authDescL[ix]
   if t<>''  then do
      frm.TxtWho.setForeground(color.black)
      frm.TxtWho.setText(t)
   end
   else do
      frm.TxtWho.setForeground(color.red)
      frm.TxtWho.setText('More information about' -
                          frm.LstWho.getSelectedItem()-
                         'is not available')
   end

--------- This class handles Item Events for all List Boxes
class AboutClassLstItem implements ItemListener
 Properties inheritable
   frm         = AboutFrame -- frm is an object of class AboutFrame
   myEventName  = String  --a string is passed and available in the class

 method AboutClassLstItem(x = AboutFrame, anEvent = String)
   frm         = x

 method itemStateChanged(e = ItemEvent)
  -- Only one LstBox, so no need for an "if myEventName='lstWho' .."
  ix=frm.LstWho.getSelectedIndex()  -- Get the selected line -if any

  frm.TxtWho.setForeground(color.black)
  if ix >= 0 then frm.TxtWho.setText(frm.authDesc[ix])
            else frm.TxtWho.setText(' ')
```

Figure 16 (Part 2 of 3). AboutFrame, Event Handling with More Classes

5.6.5.3 AboutFrame, Adding a Main Method

Up to now, concentration was on the methods for users of AboutFrame. To end this chapter, an example of how AboutFrame can be called from a program is given.

The example serves yet another purpose: when starting the Java virtual machine the name of a class file to execute is passed. The Java virtual machine passes control to the Method named **main()**. Therefore, if AboutFrame must be started from the console, a “main” method must be added, as shown in Figure 17 on page 72.

```
    -- Declare a MAIN, else JAVA gives a warning and
    --                          cannot start this class as first program.
    -- At the same time we can refuse to start on VM/ESA
1  method main(args=String[])       public  Static
2     Osname = System.getProperty('os.name');
      If Osname='VM/ESA' then Say 'Java GUI is n/a on VM/ESA'
      else do
3      abf = AboutFrame(3)         /* Build our frame, 3 authors */
4      abf.SetTitle('About About') /* Set a title  */
       abf.SetApplText('Info about this NetRexx application')
       abf.SetAuthor('Kris Buelens','is the author of AboutFrame.', -
                     'He comes from Belgium and loves VM')
       abf.SetAuthor('** version **','This is version 1.2 of AboutFrame.' -
                     'Created at 27 February 1997, in Boeblingen, Germany.')
       abf.SetAuthor('** project **','AboutFrame has been written during'-
                     'a redbook project: Java and NetRexx in VM/ESA' -
                     '(double click for more info).', -
                     'The output of this project is a redbook. It explains'-
                     'how Java and NetRexx can be used in VM/ESA and' -
                     'contains some sample coding')
       abf.SelectAuthor(0)         /* Select first author */
5      abf.ShowAbout()             /* Show it  */
6      bbf = AboutFrame(1)         /* Build our frame, max 1 author */
       bbf.SetTitle('About Christiaan') /* Set a title  */
       bbf.SetAuthor('Mr C. Buelens','Alias Kris',-
                     'Comes from Belgium, loves his wife more than VM')
       bbf.ShowAbout()                  /* Show it  */
      end
```

Figure 17. AboutFrame, a Main Method

1 The signature for "main" must be as shown: an array of Java String objects. AboutFrame doesn't use the parameters.

2 This statement can serve as a usage example of the "System" class. The getProperty method makes it possible, for example, to check in which operating system the Java virtual machine is running. As VM/ESA does not support Java frames, AboutFrame gives a message and stops.

3 At this line, an AboutFrame object is created, asking for room for three authors. Variable "abf" is the handle to this object.

4 From here on, the information for the AboutFrame is completed by calling methods of the AboutFrame class. On each call the handle "abf" must be specified, only this way the methods know on which instance of the AboutFrame class they work.

5 As the information is now complete, the "abf" AboutFrame can be presented to the end-user.

6 Just for fun, and to show that the AboutFrame class can manage multiple AboutFrame objects, the program creates a second AboutFrame. The handle to it is stored in "bbf." Note that it would have been possible to intermix the calls building both AboutFrames.

5.6.6 Avoiding Empty Frames

If the constructor method of AboutFrame, shown in Figure 12 on page 64, were to include `win=Frame(″About″)`, then the interaction diagrammed in Figure 18 would occur.

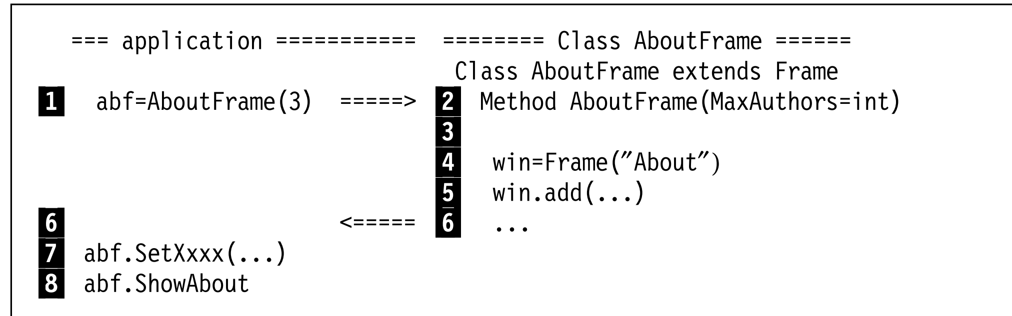


Figure 18. AboutFrame, Empty Frame Problem

1 The application calls AboutFrame′s constructor.

2 The constructor is started.

3 The class knows that AboutFrame is a Frame, and calls the constructor of the parent: Frame() to create a frame. Variable "this" points to this frame.

4 Here, yet another frame is created; "win" points to it.

5 This second window is populated with the list boxes and so on.

6 When the constructor ends, the class returns "this" to the application. Consequently its variable "abf" points to the first empty frame.

7 The application calls other methods of AboutFrame.

8 When the build process is complete, the application calls a method to show the frame pointed to by "abf." This is the empty frame.

The solution is simple: AboutFrame′s construction should not create an extra frame, but simply use the first one. This may be made explicit in the code as follows:

```
Method AboutFrame(MaxAuthors=int)
 add(...)
```

or if one wants to make it clearer that "add" is called for a frame:

```
Method AboutFrame(MaxAuthors=int)
 win = this
 win.add(...)
```

Chapter 6. Reading and Writing Files from NetRexx

6.1 Reading BFS Character Data Files

The mechanics of reading a BFS file from NetRexx include the creation of a FileReader object and a BufferedInputStream from it, and then using BufferedInputStream's read methods. We check for EOF by testing for a null returned object from readline().

Figure 19 shows how to read a BFS file.

```
/* readAfile.nrx - how to read a file from netrexx */
Parse Arg path
 If path = '' Then path = '/usr/NetRexx/nrover.doc'
 Do
   inFile = FileReader(path)
   source = BufferedReader(inFile)
 Catch r=IOException
   Say 'Can not open file' r.getMessage()
   Exit 28
 End
 Do
   Loop Forever
     line = source.readLine()
     If line = null Then Leave
     Say line
     End
   source.close()
 Catch r2=IOException
   Say 'read I/O error' r2.getMessage()
   Exit 1
 End
```

Figure 19. readAfile.nrx

6.1.1 Reading CMS Character Data Files

You can read CMS files from NetRexx by creating a BFS external link. A way to create an external link is by using the OPENVM command. For example, if you wanted to read DAVE CHILI A as pathname /home/sal/jailhouse and then run the readAfile class,

```
OPENVM CREATE EXTL /home/sal/jailhouse CMSDATA //DAVE.CHILI.A,&&&
Ready; T=0.04/0.05 13:21:39
NRR /home/sal/readAfile /home/sal/jailhouse
Calling Java program readAfile.class ....

1.0  DAVE'S FAVORITE TEXAS CHILI RECIPES
  NOTE:  These are authentic Texas recipes, so there are NO BEANS in any
  of them. You can, of course, cook up a batch of red or pinto beans to
  be served on the side.

1.1  LONE STAR CHILI
  o   8 lbs. chuck (beef), coarsely ground
  o   3 (8oz.) cans tomato sauce
  o   2 onions, chopped
  o   5 garlic cloves, finely minced
  o   Chili powder - lots of it
  o   Cumin to taste
  o   Oregano to taste
  o   Salt to taste
  o   Masa
  Method: Brown beef in an iron skillet and transfer to chili pot. Add
  tomato sauce and equal amount of water. Add onions, garlic and chili
  powder. Simmer for 20 minutes. Add cumin, oregano, and salt to taste.
  Simmer covered for 30 minutes to an hour. Add masa to achieve desired
  thickness. Cook 10 additional minutes. Correct seasonings to taste.

1.2  DALLAS JAILHOUSE CHILI
  o   1/2 (1/2) cup olive oil
  o   2 lbs. coarsely ground beef
  o   2 cloves garlic, minced
  o   1 & 1/2 (1 1/2) Tbs. paprika
  o   1 Tbs. comino seeds
  o   3 Tbs. chili powder
  o   1 Tbs. salt
  o   1 tsp. white pepper
  o   3 cups water
  Method: Heat oil. Add meat, garlic and seasonings. Cover and cook slowly
  for 4 hours, stirring occasionally. Add water and continue cooking until
  slightly thickened, about 1 more hour. Legend holds that this chili was
  so good that the good guys turned bad just to get thrown in jail for a
  taste of it!

1.3  UNITS OF MEASUREMENT
  o   lb. (lbs.)  pound; 1 pound = 2.2 kilograms
  o   tsp.  teaspoon
  o   Tbs.  tablespoon

Ready; T=1.40/1.67 13:22:58
```

From then on (or until the link is removed by using the OPENVM ERASE command, for example) you can use the pathname /home/sal/jailhouse to refer to the DAVE CHILI A file just like any other BFS pathname.

Notes:

1. The filemode defaults to * if not specified.
2. The path (directory) part of the pathname must exist.
3. You can specify read/write access by using "&&&" after the name.
4. The file must be on an accessed CMS minidisk or SFS directory.

6.2 Reading from the console

Use the NetRexx ASK statement to read console input.

```
Say "Enter 0 or 1"
answer=Ask
Say answer
```

6.2.1 Useful Control Sequences

The are several control sequences that can be use while your NetRexx program is running. All the control sequences start with the cent sign (¢), X'4A'. If your keyboard has no cent sign key, include the following in your PROFILE EXEC to set a PFKey to enter it for you. For example, to set PF2 to generate the control sequence introducer:

```
/* set pf2 as cent sign */
"CP TERM LINEDEL OFF"
"CP SET PF2 DELAY" '4A'X
```

Note: By default the cent sign is CP's "line delete" character. So to be able to use these control sequences one must change CP's LINEDEL setting. As on 3270 terminals LINEDEL (and CHARDEL) are not very useful, we recommend turning it off. This can be done on a system wide level in the SYSTEM CONFIG file (usually on MAINT CF1) or on a virtual machine level using CP's TERMINAL command.

Table 2. Useful control sequences

Control Character	ASCII Control Sequence	OpenEdition Sequence
<backspace>	control-H	¢h
<carriage-return>	control-M	¢m
<EOT>	control-D	¢d
<EXT>	control-C	¢c
<NAK>	control-U	¢u
<tab>	control-I	¢i
<SYN>	control-V	¢v

Hints:

1. Use <ETO> <ETO> to simulate end of file.
2. Try <SYN> to terminate a process (SIGQUIT).
3. Try <EXT> to kill a running process (SIGINT).

The following sequence creates two input lines after the ENTER key is pressed:

```
Line one¢mLine two
```

Remember

To cancel a running Java program, enter **¢c**. It is less disruptive than #CP IPL or HX.

For more information on the escape sequences, see *OpenEdition for VM/ESA User's Guide*, SC24-5727.

6.3 Writing BFS Character Data Files

The usual way to write a file from NetRexx is by creating a FileWriter object and then creating a PrintWriter or BufferedWriter class from it. If you want to output textual representations of primitive values and objects you should use the print methods of the PrintWriter class. To write a Java String you can use the BufferedWriter.write() method.

Figure 20 shows a copy program that read and writes text lines:

```
/* writeAfile.nrx - a simple copyfile program */
Parse Arg inPath outPath .
 If inPath  = ''  Then Do
   inPath  = '/usr/NetRexx/nrover.doc'
   outPath = './my.nrover.doc'
   End
 Say 'input.:' inPath
 Say 'output:' outPath
Do
 inFile = FileReader(inPath)
 source = BufferedReader(inFile)
 If outPath = '' Then outFile = OutputStreamWriter(System.out);
 Else outFile = FileWriter(outPath)
 dest   = BufferedWriter(outFile)
 Loop Forever
   text = String source.readLIne()
   If text = null Then Leave
   dest.write(text,0,text.length())
   dest.newline()
   dest.flush()
   End
 source.close()
 dest.close()
Catch io=IOException
 Say 'IOException' io.getMessage()
End
```

Figure 20. writeAfile.nrx

6.3.1 Writing CMS Character Data Files

Just as you have to create an external link to read a CMS file, you must also create an external link to write a CMS file. An external link that will be used for output should explicitly include the appropriate access mode string and file attributes For example, to create a file called QTIME TEMP A using the writeAfile class, one needs to create a BFS external link to that CMS file first. Let's call it /home/dave/qtime.f; then, using OPENVM:

```
openvm create extl /home/dave/qtime.f cmsdata //QTIME.TEMP.A,&&&,
                                     recfm=v,lrecl=80
```

After that, one can run the writeAfile class with /home/dave/qtime.f as the output file to create the QTIME TEMP A file:

```
nrr writeAfile /usr/NetRexx/qtime.nrx  /home/dave/qtime.f
```

6.4 Working With Binary Files

There are four classes that are used in conjuction with primitive data types, DataInputStream, DataOutputStream, RandomAccessFile, and PrintWriter.

Use DataInputStream and DataOutputStream to read and write integers, floats, characters, bytes, and UTF strings. RandomAccessFile is a combination of the two, but with the ability to seek. It is not a member of the Stream class family. PrintWriter allows you to display primitive data types in a textual manner.

Figure 21 shows a program that creates a file of factorial numbers, then reads the file backwards and prints the numbers just read.

```
/* factTable.nrx - write and read factorial integers*/
 table = './factorial.table'
 outFile = FileOutputStream(table)
 dest = DataOutputStream(outFile)
 Say 'Creating factorial table' table
 iterations = 10
 fact = int 1
 Loop i=1 to iterations
   fact = fact * i
   dest.writeInt(fact)
   Say fact
   Catch IOException
     Say 'got IOException - writing'
   End
 dest.close()
 source = RandomAccessFile(table,'r')
 printFile = PrintWriter(System.out);
 printFile.println("Reading factorial table backwards")
 intSize = 4                            -- a Java integer is 4 bytes long
 offset  = (iterations * intSize) - intSize; -- point to last int
 Loop While offset > -1
   source.seek(offset)              -- re position file pointer
   fact = source.readInt()          -- read the integer
   printFile.println(fact)          -- PrintWriter.println(int)
   printFile.flush()
   offset = offset - intSize        -- move backwards by one int size
   Catch IOException
     Say 'got IOException - reading'
   End
Do
 source.close()
 printFile.close()
Catch c=IOException
 Say 'IOException' c.getMessage()
End
```

Figure 21. factTable.nrx

Chapter 7. Code Pages - ASCII <> EBCDIC Issues

7.1 History, Experience

One of the first attempts during the project was making a Java client and server "talk" to each other.

We used two simple examples from the OS/2 NetRexx redbook *Creating Java Applications Using NetRexx*, SG24-2216. The client program is shown in Figure 22. The server program is shown in Figure 23 on page 82.

```
/* Client HTTP program, sends a request to an HTTP server:
   Usage: Java ClntSock <server> <portnumber> <requeststring> */

parse arg server port str              -- capture + test arguments
if server='' | port='' then
  do
    say 'Usage: java ClntSock <server> <portnumber> <requeststring>'
    exit 1
  end

parse str get rest                     -- check requeststring
if get <> '' then do
   if get <> 'GET' then do
     say 'Request string must be: GET /http-page'
     exit 8
   end
   str = 'GET' rest                    -- make get uppercase
end

do                                     -- ready to process
  say 'Connecting to server:' server '(port:' port')'
  mysocket = Socket(server, port)      -- actual connect
  say 'Requesting:' str                -- what we want
  say
                                       -- output: send
  out = PrintWriter(OutputStreamWriter(mysocket.getOutputStream()))
                                       -- input: receive
  in  = BufferedReader(InputStreamReader(mysocket.getInputStream()))
  out.printLn(str)                     -- send our requeststring
  out.flush()                          -- needed on some platforms
  say 'Response:'
  line = String(in.readLine())         -- read response
  loop while line \= null
     say ' ' line                      -- print it out
     line = in.readLine()
  end
catch e=IOException
  say 'IOException (' e ') caught:\n' e.getMessage()
end
```

Figure 22. Socket Client Program

© Copyright IBM Corp. 1998

```
/* Server HTTP program, accepts a request from an HTTP client:
   Usage: Java SrvSock <portnumber>                            */
do
  if arg = ''  then arg = 80          -- default port
  serverS = ServerSocket(arg)         -- register at port: server socket
  say 'Server:' serverS
  loop forever
    serviceS = serverS.accept()       -- listen/accept client: service socket
    say serviceS '\n  connected at:' Date()
    ptrW = PrintWriter(OutputStreamWriter(serviceS.getOutputStream()))
    sIS  = BufferedReader(InputStreamReader(serviceS.getInputStream()))
    loop while sIS.ready()            -- consume HTTP request
      line = String(sIS.readLine())
    end
    filename = 'SrvSock.nrx'          -- always returning the source file
    fileBR = BufferedReader(FileReader(filename))
    line = String(fileBR.readLine())
    loop while(line <> null)          -- add lines of source file
       ptrW.printLn(line)
       line = fileBR.readLine()
    end                               -- end loop while(line <> null)
    ptrW.close()                      -- close output and socket
    serviceS.close()
  catch e=IOException
    say 'IOException caught:' e.getMessage() -- error message
  end                                 -- end loop forever
end
```

Figure 23. Socket Server Program

When the server and the client both run on VM, everything works fine.

During another test, we started the server in a PC and ran the client on VM. This no longer worked well. The server and the client didn't understand each other: when the client sent its GET request, the server just hung.

We figured out that we could avoid the hang by appending X'0D25' to the GET request sent from VM (X'0D25' is the EBCDIC equivalent of CR/LF). But then the display of the data sent back by the server was unreadable. Only after translating the VM console from ASCII to EBCDIC did the display start making sense.

This is how we discovered the ASCII - EBCDIC issues that originate when running Java on VM/ESA or OS/390.

7.2 Background Information - Codepages

When using Java on VM, you should be surprised it works, because:

- The S/390 world, where VM is running, is using EBCDIC. This means, for example, that X'81' displays as the letter A, and X'F1' is the character 1.
- The internet, where Java feels at home, is using ASCII. This means that X'31' displays as the character 1 and X'41' is an A.

Even though these character sets are different, Java and NetRexx work fine on VM. Java and NetRexx source files are stored in the BFS in EBCDIC format. The class files, however, are stored in ASCII.

It can work because Java 1.1 has what is called **Internationalization** support.

7.3 Internationalization

With this support, platforms running Java can define a **codepage**. A codepage defines how hexadecimal values are displayed.

Even in EBCDIC, different code pages do exist. For example in Belgium, where codepage 500 is being used, X′4F′ displays as an exclamation point (!). In the US, and most other countries, X′4F′ displays as a vertical bar (|).

With the internationalization support, Java defines a whole new set of methods to read and write. They allow the specification of a codepage to use on the stream one is reading from or writing to.

This new IO support is fully described on the internet:

http://java.sun.com/products/jdk/1.1/docs/guide/io/index.html

Internationalization means more to Java than just codepages. Other examples are date and number formats. A good starting point to learn about the internationalization can be found on the internet:

http://java.sun.com/products/jdk/1.1/docs/guide/intl/index.html

7.3.1 Streams?

Readers without some experience in the workstation world are probably not familiar with the term **stream**.

In the S/390 environment, programs read records from files. A record is just a piece of a file, and the filesystem keeps track of the length of each record.

In the Unix and PC environment, records are not defined. A file simply is a series (or a stream) of bytes; record boundaries are not really defined. A program can divide the file in elements (or records) any way it likes. Often the combination of CR/LF (CarriageReturn/LineFeed) is recognized as record boundaries.

So in the Unix and PC programming environment, one often uses the term “stream” to refer to a file. As the same methods are used to read/write to the TCP/IP network, the term “stream” is a better choice than “file.”

7.3.2 Java IO Support

Prior to Java 1.1, Java knew only **byte streams**, to which no codepage is defined. The inputStream and outputStream classes and their subclasses defined the methods to use these streams.

With the internationalization, Java introduces the term **character stream**. A character stream is externally similar to a byte stream, but internally contains 16-bit Unicode characters instead of 8-bit characters. The 16-bit Unicode allows us to represent almost any character. The classes Reader and Writer and their subclasses define the methods to use these character streams.

The InputStreamReader and OutputStreamWriter classes can act as a bridge between byte streams and character streams. They allow the specification of a codepage to convert between byte and character streams.

7.4 VM Java Codepage

The Java implementation on VM/ESA and OS/390 uses codepage 1047 as default. As 1047 is an EBCDIC codepage, Java programs reading "normal" VM files and displaying them on a virtual console work fine.

However, if a Java program uses VM data (from a file or from the console) and sends it to the network, the client is probably unhappy as it is expecting to get ASCII encoded data. Similarly when a VM program reads from the network, it probably gets ASCII encoded data. When this data is to be used on VM some translation must be done.

You can use the System class to find the default codepage set by the Java virtual machine. This small NetRexx program will display them all:

```
p = Properties System.getProperties();
p.list(System.out)
```

A direct reference to the default codepage can be made by:

```
cp  = System.getProperty('file.encoding')
Say 'we use codepage' cp
```

7.4.1 Solution for Client Server Programs

The solution which allows client and servers to communicate is using ASCII on anything that is sent to the network. The InputStreamReader and OutputStreamWriter classes can be used for that.

To make the sample client and server work, only two lines in the client program have to be changed:

```
out = PrintWriter(OutputStreamWriter(mysocket.getOutputStream(), -
                                     '8859_1'))
                                     -- input: receive
inStream= InputStreamReader(mysocket.getInputStream(),'8859_1')
```

For the server program the changes are minor, too:

```
  ptrW = PrintWriter(OutputStreamWriter(serviceS.getOutputStream(),-
                                     '8859_1'))
  sIS  = BufferedReader(InputStreamReader(serviceS.getInputStream(),-
                                     '8859_1'))

  fileBR = BufferedReader(InputStreamReader(FileReader(filename),'Cp1047'))
```

Note: In the last line above we explicitly set the codepage to "Cp1047" for the file being read. But as "Cp1047" is VM's default, that change is not required when the server runs on VM. If, however, the server would run on an IBM Network Station and read a VM EBCDIC file, specifying "Cp1047" is a must.

Java’s character stream classes define a method which allows you to find out which codepage has been defined (or defaulted) for the stream:

```
say 'Encoding of "instream" is' inStream.GetEncoding()
```

Remember

Every Java program running on VM/ESA or OS/390 must make sure it sends ASCII encoded data to the network, because the internet network is an ASCII world.

When installing an existing Java networking program on VM/ESA or OS/390, you may have to apply a few changes just to make it use ASCII for its network interfaces.

7.5 IBM Network Station and Codepages

Users of IBM Network Station may want to refer to 11.8.1.2, “Codepages - ASCII <> EBCDIC for Network Stations” on page 129 for some important items.

Chapter 8. TCP/IP Networking

8.1 Translating between EBCDIC and ASCII

As mentioned previously in this redbook (see 7.4, “VM Java Codepage” on page 84), by default input and output streams in the VM/ESA implementation of Java assume codepage CP1047 (EBCDIC). Most TCP/IP servers use code page 8859 (ASCII). If you take a Java or NetRexx TCP/IP client or server off the shelf there is a chance that it might not work on VM/ESA.

This problem can be solved if you have access to the NetRexx or Java source code. Solving the problem includes instantiating the correct Java classes to do the translation between ASCII and EBCDIC.

The key classes are InputStreamReader and OutputStreamReader. When instantiating the classes use ″8859_1″ as the encoding string to get a stream that talks ASCII to the outside world. The following NetRexx sample shows you how to do it:

```
connection = Socket(host,port)
asciiIn = InputStreamReader( connection.getInputStream() ,'8859_1')
asciiOut= OutputStreamWriter(connection.getOutputStream(),'8859_1')
fromNet = BufferedReader(asciiIn)
toNet = BufferedWriter(asciiOut)
```

8.1.1 readLine() and printLn()

Many Java Networking programs written prior to the JDK 1.1 use the methods DataInputStream.readLine() and PrintStream.printLn() to exchange text lines. Both methods display their shortcomings when run in a multi-platform environment.

DataInputStream.readline() has been deprecated and PrintStream.printLn() was superseded in JDK 1.1. Java programs utilizing these methods should be modified in order to operate correctly in a CMS environment. The standard networking boiler plate used prior to JDK 1.1:

```
InputStream inStream = socket.getInputStream();
DataInputStream in = new DataInputStream(inStream);
OutputStream outStream = socket.getOutputStream();
PrintStream out = new PrintStream(outStream);
```

can be changed as follows in order to operate correctly in the CMS environment:

```
InputStream inStream  = socket.getInputStream();
InputStreamReader in = new InputStreamReader(inStream,'8859_1')
OutputStream outStream = socket.getOutputStream();
OutputStreamWriter outWriter = new OutputStreamWriter(outStream,'8859_1');
PrintWriter out = new PrintWriter(outWriter);
```

readUTF and writeUTF

You should not have any difficulties if your Java client and server use the DataInputStream.readUTF and DataOutputStream.writeUTF methods to exchange Java strings.

8.2 Simple TCP/IP Client

Figure 24 shows a simple implementation of a NetRexx client that exchanges text lines with an ASCII server.

```
/* Client.nrx                 - simple ascii client */
Parse Arg host port

If host='' | port='' Then Do
  Say "Usage error. Client <host> <port>"
  Exit 27
  End
Do
  connection = Socket(host,port)
  asciiIn = InputStreamReader( connection.getInputStream() ,'8859_1')
  asciiOut= OutputStreamWriter(connection.getOutputStream(),'8859_1')
  fromNet = BufferedReader(asciiIn)
  toNet = PrintWriter(asciiOut)
  Loop Forever
    Say "Enter request or quit :"
    request=Ask
    If request = 'quit' Then Leave
    toNet.printLn(request)
    toNet.flush()
    response = fromNet.readLine()
    If response = null Then Leave
    Say 'Response:' response
    End
  connection.close();
Catch netEx=IOException
  Say "Got some kind of Error:" netEx.getMessage()
End
```

Figure 24. Simple NetRexx TCP/IP Client

8.3 Simple TCP/IP Server

Figure 25 on page 89 shows the server code familiar to many UNIX/Networking Java programmers. It tries to simulate the fork()/exec() paradigm, now translated to NetRexx.

```
/* TCPServer.nrx */
import java.net.
import java.io.

1 Class TCPServer public implements Runnable,Cloneable
  Properties Inheritable
    runner = Thread null
    server  = ServerSocket null
    data   = Socket null
  Method TCPServer()
    super()
2 Method StartServer(port=int) Public Signals IOException Protect
    If runner = null Then Do
      server = ServerSocket ServerSocket(port)
      runner = Thread Thread(this)
3      runner.start()
      End
  Method StopServer() public Protect
    If server <> null Then runner.stop()

4 Method run()
    If server <> null Then Do
5        Loop Forever
          datasocket = Socket server.accept()
          newSocket  = (TCPServer clone())
          newSocket.server = null
          newSocket.data = datasocket
          newSocket.runner= Thread Thread()
6         newSocket.runner.start()
        Catch se=Exception
          Say "While creating new client th -ex:"se.getMessage()
        End
      End
    Else
7     run(data)
```

Figure 25 (Part 1 of 2). TCP/IP Server in NetRexx

```
8 Method run(s_=Socket)
    Say "New Client:" s_
    Do
      sIn = BufferedReader(-
              InputStreamReader(s_.getInputStream(),"8859_1"))
      sOut= PrintWriter(-
              OutputStreamWriter(s_.getOutputStream(),"8859_1"))
    Catch se=IOException
     Say "Error while opening Client Socket" se.getMessage()
    End
9   runUser(sIn,sOut)
    Do
      sOut.close()
      sIn.close()
    Catch ce=IOException
      Say "Error while closing client" ce.getMessage()
    End
10 Method runUser(sIn_=BufferedReader,sOut_=PrintWriter)
     Do
       l = sIn_.readLine();
       If l = null Then Return
       sOut_.printLn("echo:" l)
       sOut_.flush();
     Catch rwe=IOException
       Say "Error while reading/writing" rwe.getMessage()
     Finally
       Say "end of Client job"
     End
```

Figure 25 (Part 2 of 2). TCP/IP Server in NetRexx

1 The TCPServer class implements the Runnable interface so it can be run as a thread. It also uses the Cloneable interface to make copies of itself.

2 After the TCPServer class is instantiated, the StartServer method gets called to get things started. If not a client thread, it creates a new ServerSocket to listen at a port number and a new thread with the current object's data and methods, then **3** starts the new thread by calling the Thread.start() method.

4 run() is called when the new thread is up and running, ready to do work.

5 If it is the server thread, loop waiting for new clients. For each new client, clone the current object (using the Object's default method), so it can be used as a new client thread, set some properties for a client thread, create a new thread and **6** start the newly created thread.

4 run() gets called for this client thread. Since this is not the server thread, **7** run(data) is called.

8 run(Socket) receives control. Note that it has the same name as run(), but it has a different signature. This is called overloading. Using the Socket reference, ASCII input and output streams are created and **9** runUser() is called.

10 runUser() is a simple echo program that echos the client's request and terminates the client thread.

8.3.1 Extending the Server

A nice thing about Object Oriented programming is that you can modify the methods of a class to behave the way you want. This is called extending a class. On the TCPServer class, the run(Socket) or runUser() method can be overridden to work the way you want.

In all cases you will need to override the runUser() method to make the server perform useful work. You may also want to override the run(Socket) method if you need to use other codepages or other Stream classes. Figure 26 shows you how to override the runUser() method by extending the TCPServer class.

```
import java.net.
import java.io.
1 class ServerHandler public extends TCPServer
2   Method runUser(input=BufferedReader,output=PrintWriter)
     Loop Forever
       request = input.readLine();
       If request = null Then Leave
       Select
         When request = 'QUIT' Then Leave
         When request = 'TIME' Then Do
           p=CMSPipe('cp q time|take 1|fitting *>java')
           answer=p.readto()
           p.endPipe()
           If answer = null Then answer='could not get time'
           End
         Otherwise
           answer = "Invalid request"
         End
       output.printLn(answer)
       output.flush();
     Catch IOException
       Say 'got error while reading/writing'
     End
```

Figure 26. ServerHandler.nrx, Extending the TCPServer.nrx Class

1 The extends keyword tells NetRexx that your ServerHandler class will inherit all of the public or inheritable variables and methods of the TCPServer class.

2 The ServerHandler.runUser method will override the old method in the superclass TCPServer.runUser. Basically, now your ServerHandler class is the TCPServer with an updated runUser() method.

8.3.2 Starting the Server

To start the server you need a simple NetRexx script or class to create a new instance of ServerHandler and then invoke ServerHandler.startServer(port).

```
/*--------------------------------------------------------------*/
/* This NetRexx program starts a multithreading TCP/IP server    */
/*--------------------------------------------------------------*/
Parse Arg port
 If port = '' Then Do
   Say 'Usage:  myserver <port> '
   Exit 27
   End
```

```
serv = ServerHandler ServerHandler()
serv.startServer(port)
```

Note: If you get the following NetRexx error message while extending a class:

Error: Checked exception ...

then NetRexx is telling you that you are not catching all the exceptions that you should catch. This may be because the superclass calling that method does not know how to handle a new exception that might be thrown by your code.

Chapter 9. Java and CMS

As long as all software is not written in Java there will be a need to interface with it. Java offers two ways to communicate with non-Java applications: the Runtime.exec() method and the Java Native Interface (JNI). We chose to write our own interfaces for more convenient access to CMS services.

9.1 Executing non-Java Programs

9.1.1 Using Runtime.exec()

It is possible to run other OpenEdition programs from inside the Java virtual machine. Each Java virtual machine has a Runtime class associated with it. You can call the static Runtime.getRuntime method to get a reference to the current Runtime class. The exec() method of the Runtime class can then be used to create a new Java Process class.

The Process class describes the program running externally to the Java interpreter. The Process class has many methods to manipulate the running program. You cannot directly create a Process class; instead, you must ask Java to create one for you by calling the Runtime.exec method.

The cook book method of starting an external program from inside the Java virtual machine is as follows:

```
/* */
r = Runtime Runtime.getRuntime()    -- get a reference to runtime
p = Process null                    -- create a new process reference
do
  p = r.exec("/bin/ls")             -- call exec to get a new process
  p.waitFor();                      -- use the Process waitFor method
                                    -- to wait for the program to end
catch Exception                     -- got here if an error
  Say "could not start program"
end
```

Another feature of the Runtime.exec method is that its redirects the stdin, stdout, and stderr streams of the running program. Every C program has them: stdin to read from the terminal, stdout to type to the terminal, stderrr to type out error messages. You can use the getInputStream(), getOutputStream(), and getErrorStream() methods of the Process class to get at them.

Changing the previous example so that we can see the results of the /bin/ls command yields the following:

```
/* */
r = Runtime Runtime.getRuntime()    -- get a reference to runtime
p = Process null                    -- create a new process reference
do
  p= r.exec("/bin/ls")              -- call exec to get a new process
  iS = InputStreamReader(p.getInputStream()) -- get stdout
  in = BufferedReader(iS)                    -- buffered
  Loop Forever
    list = in.readLine()            --   read next line from stdout
    If list = null Then Leave       --   done? - leave
```

```
    Say 'File:' list
    End
  p.waitFor();                    -- use the Process waitFor method
  rc=p.exitValue()                -- let's get the return code and
  Say "Rc="rc                     -- display it
catch Exception                   -- got here if an error
  Say "could not start program"
end
```

Note - Cp1047 translation under the covers - Stream Classes

By default, Java streams will translate data to or from the default codepage while exchanging data with your program. If you want to exchange binary data, use the DataInputStream or DataOutputStream classes. Otherwise, if you use any of the other Stream classes, you are setting yourself up for a nice surprise. Take a look at the following NetRexx code:

```
p= r.exec("Cprogram")
out= OutputStreamWriter(p.getOutputStream())
Loop i=0 to 5                                  --write six numbers 0-5
  out.write(int i)
  End
```

Subsequent reads from a C or Assembler program, for example using getc(), will result in:

'00'x '01'x '02'x '03'x '37'x '2D'x

This is because X'04' is ASCII <EOT>, which gets translated to X'37', EBCDIC <ETO>; similarly, X'05' is ASCII <ENQ>, which gets translated to X'2D', EBCDIC <ENQ>.

Similar results can be expected if you write to a file or a network socket.

9.1.2 Using JNI

Another way to interface Java to the outside world would be to use the JNI API, which involves writing a C DLL program. The JNI API was not explored during this residency.

9.2 The cms.util Package

We created a NetRexx package called cms.util that allows you to run CMS execs and CMS pipelines from NetRexx and Java programs. You must use the *import cms.util.* NetRexx statement in your NetRexx program and have the correct CLASSPATH environment variable. The installation script JNRCMS LOADBFS, supplied with the JNRCMS PACKAGE that is included with the sample program set for this redbook (see 2.7.1.5, "Download and Install the SG245148 Package" on page 19), will set the correct CLASSPATH and directory structure. To utilize the classes, just add the import statement at the beginning of your program (do not forget the trailing **dot**); for example:

```
/* example.nrx */
import cms.util.
⋮
p=CMSPipe( ... )
⋮
```

In the following sections we describe the facilities provided in the cms.util package.

9.3 Running CMS Execs

The beauty of OpenEdition for VM/ESA is that it is still CMS, and you can use its underlying facilities to get your work done. Calling a REXX exec is as simple as writing a C program that calls the DMSCCE CSL routine. The CMSRexx class and its support module JCMSREXX module accomplish this.

9.3.1 The CMSRexx Class

Use the CMSRexx class to invoke a REXX exec from a NetRexx application.

```
Class CMSRexx Public
-- constructor
   Method CMSRexx() Public
-- instance methods
   Method exec( nameArgs=String ) Public Returns String
   Method exec( execName=String,Args=String ) Public Returns String
   Method getRc() Public Returns int
```

1. Constructor

   CMSRexx() Creates a new CMSRexx object

2. Methods

   exec(nameArgs=String) Public Returns String Executes the CMS REXX exec specified as the first token of nameArgs string and treats the succeeding tokens as the exec's arguments

   exec(execName=String,execParms=String) Public Returns String Executes the CMS REXX exec specified in execName with the string Args as the exec's arguments

   getRc() Returns int Gets the return code from the last REXX exec executed

9.3.1.1 Usage Notes

1. The CMSRexx.exec method has a limit on the size of its returned string, 8K. A return code of 200 will be set if larger.
2. The CMSRexx.exec method has a limit on the size of its input string, nameArgs (or execNamec plus execParms); it can not be larger than 8K.
3. Once the CMSRexx object has been instantiated (created), the exec method can be called multiple times with different execs and arguments.
4. The getRc method returns the return code received after a call to DMSCCE. See the *VM/ESA CMS Application Development Reference*, SC24-5761, for details.

9.3.1.2 Example

The following example creates a CMSRexx object called cmd and calls a CMS exec JTEST$1 by invoking the CMSRexx's exec method twice. The first time it uses a string with the exec's name and arguments concatenated together, and the second time they are specified as separate parameters. JTEST$1 exec simply returns the reverse of its argument.

```
/* testCMSRexx.nrx */
import cms.util.                       -- import the classes
cmd = CMSRexx()                        -- create the CMSRexx obj
parm = "Kika the cat!"
Say "calling JTEST$1 with parm" parm
result = cmd.exec("JTEST$1" parm);     -- call method exec
Say "Result:" result
--do it again
parm = "Sam the doggy!"
fn = "JTEST$1"
Say "calling" fn "with parm" parm
result = cmd.exec(fn,parm);            -- call method exec again
Say "Result:" result
exit

/* JTEST$1 EXEC */
Parse Arg parm
Return (reverse(parm))
```

9.4 Running CMS Pipelines with NetRexx

As a CMS developer, sooner or later you will want to run a CMS Pipeline with your NetRexx program. Also, the ability to read or write a single pipeline record at a time from a NetRexx application is highly attractive.

The prototype CMSPipe class tries to accomplish this. In order to read or write a record at a time from NetRexx, one needs to be able to suspend the currently running pipeline. Thanks to John Hartmann's foresight, this is possible by creating a Co-pipe as described in his paper "PIPE Command Programming Interface," available from the Pipelines Run Time Distribution Web page at URL http://pucc.princeton.edu/~ pipeline/. We implemented this as a cooperative program using the facilities of the Java Runtime class described previously in 9.1.1, "Using Runtime.exec()" on page 93, hence some of its limitations. A better implementation should be possible using the Java JNI API, but we did not attempt to explore the use of the JNI during this residency.

The way that a PIPE exchanges data with the NetRexx application is via a fitting stage. From the previously mentioned paper: "A fitting stage is the space warp through which records move between the pipeline and the application program. A fitting stage is the application's agent in the pipeline."

Two fitting stages are recognized by the CMSPipe class: *fitting *>java* to read records from the pipeline and *fitting *<java* to write records to the pipeline.

9.4.1 fitting *>java

Use the *fitting *>java* stage to consume records from the pipeline. It should be the end point of the pipeline specification. CMSPipe.readto() returns the next record in the pipeline as a Java String.

9.4.2 fitting *<java

Use the *fitting *<java* stage to produce records for the pipeline. It should be the first stage in the pipeline specification. CMSPipe.output() writes its Java String argument to the pipeline.

9.4.3 The CMSPipe Class

Use the CMSPipe class to create a CMS Pipeline and run it as a co-pipeline from a NetRexx application.

```
Class CMSPipe Public
-- constructor
   Method CMSPipe( pipeSpecification=String ) Public
-- instance methods
   Method readto() Public Returns String
   Method getReadtoRc() Public returns int
   Method severInput() Public
   Method output( record=String ) Public Returns int
   Method getOutputRc() Public returns int
   Method severOutput() Public
   Method endPipe() Public
   Method getPipeRc() Public returns int
   Method killPipe() Public
```

1 Constructor

CMSPipe(pipeSpecification=String) Public
Creates a new CMSPipe object with the pipeline specification you want to run. You can use the special fitting stages *fitting *<java* and *fitting*>java* to input and output records to the pipeline respectively.

2 Methods

readto(nameArgs=String) Public Returns String
Reads data from the pipeline if you specified the *fitting *>java* stage as the end point of your pipeline specification. The pipeline record is returned as a Java string. If EOF a null String object is returned.

getReadtoRc() Public returns int
Returns the return code set after the last readto call. It is set to 12 if EOF, to 0 otherwise.

severInput() Public
Indicates that the NetRexx/Java application does not wish to process further data from the pipeline. Any subsequent calls to readto result in a null string returned, and getReadtoRc returns 12.

output(record=String) Public Returns int
Writes out data to the pipeline if you specified the *fitting *<java* as the first stage in the pipeline specification. The Java string is sent as record to the pipeline. A return code is returned, set to 12 if EOF and 0 otherwise.

getOutputRc() Public returns int
Returns the return code set after the last output call. It is set to 12 if EOF and 0 otherwise.

severOutput() Public
: Indicates that the NetRexx/Java application does not wish to produce further data for the pipeline. Any calls to output result in a return code of 12, and getReadtoRc returns 12.

endPipe() Public
: Sets both input/output streams to EOF and terminates the pipeline.

getPipeRc() Public returns int
: Returns the pipeline's return code after it has ended. If the pipeline is still running, a zero is returned.

killPipe() Public
: Terminates the pipeline. EOF on input/output are not set. Data may be lost.

9.4.3.1 Usage Notes

1. Two special pipeline stages are used to transfer data between the NetRexx application and the CMS Co-pipe. The *fitting *>java* stage is used to read data from the pipeline and the *fitting *<java* stage is used to write data to the pipeline.
2. To read data from the pipeline, use the *fitting *>java* stage as the last stage in the pipeline. The valid methods for this kind of pipeline are: readto, getReadtoRc, severInput, endPipe, killPipe, and getPipeRc.
3. To write data to the pipeline, use the *fitting *<java* stage as the first stage in the pipeline. The valid methods for this kind of pipeline are: output, getOutputRc, severOutput, endPipe, killPipe, and getPipeRc.
4. Do not transfer non-character data as it will be translated by Java.
5. You can run pipeline specifications without any Java fitting stages. The pipeline will be run to completion when the CMSPipe constructor is called. The only valid methods for this kind of pipeline are endPipe() and getPipeRc().
6. Once the CMSPipe is used up, in order to run another pipeline you must create a new CMSPipe object. You can reuse the CMSPipe reference after a call to endPipe().

   Correct:

   ```
    p=CMSPipe(first pipe)
   :
    p.endPipe() p=CMSPipe(second
   pipe)
   ```

   Incorrect:

   ```
   p=CMSPipe(first pipe)
   :
   p=CMSPipe(second pipe)  -- lost reference to first pipe,
                           -- pipeline in limbo
   ```

7. You should always use the endPipe() method to terminate the pipeline; otherwise you may have some memory leakage.
8. After the endPipe() method you can safely reuse the CMSPipe's object reference.
9. You will get a return code of 65 if you forget to type *fitting* and have only **<java* or **>java*.
10. If you use the incorrect methods for the fitting that you specified (for example, readto with *fitting *<java*), you will get a return code of 12.

11 An invalid pipeline specification will set a non-zero return code after the CMSPipe constructor is executed, and the Co-pipe will terminate.

12 For more information about Co-pipes, refer to John Hartmann's paper, "PIPE Command Programming Interface," cited above.

9.4.3.2 Examples

1. Reading data from a pipeline. The following example reads the results of the CP QUERY NAMES command:

```
/* jexample1.nrx */
import cms.util.                -- import the CMSPipe.class
mypipe = CMSPipe("cp q names| split ,| fitting *>java")
1 rc=mypipe.getPipeRc()
if rc <> 0 Then Exit rc
Loop Forever
  line = mypipe.readto()
2   If line = null Then Leave
  Say line
  End
3 rc=mypipe.getPipeRc()
Say 'The CMS pipeline ended with retcode' rc
mypipe.endPipe()
```

1 This statement is used to test if the pipeline has been started correctly. Literally, getPipeRc() can only return the pipeline's returncode after the pipeline has ended. But if the returncode is already non-zero now, it means that the pipeline didn't even start. There must have been a syntax error in the pipeline's specification.

2 This is a Java/NetRexx way of testing if a line was read from the pipeline. Programmers used to coding pipeline subroutines in REXX would maybe code:

```
if mypipe.getReadtoRc() <> 0 then leave
```

3 At this time, the pipeline has ended and we can obtain the returncode of the whole pipeline. In our very simple example the returncode obtained here can only be zero.

2. Writing data to a pipeline. The following example writes a couple of Java Strings to a CMS file:

```
/* jexample2.nrx */
import cms.util.            -- import the CMSPipe.class
pObj = CMSPipe("fitting *<java | xlate | > $TEMP $JLINES A")
rc=pObj.getPipeRc()
If rc <> 0 Then Exit rc
Loop i=1 to 10
  line = 'This is line number:' i
  rc=pObj.output(line)
  If rc <> 0 Then Leave
  End
pObj.endPipe()
```

Note that this illustrates an alternative to the use of external links (described previously in 6.1.1, "Reading CMS Character Data Files" on page 75 and 6.3.1, "Writing CMS Character Data Files" on page 78) for accessing CMS minidisk or SFS files from Java or NetRexx programs.

For some more usage examples, refer to 10.4, "GUIMON - the Client-Server Communication" on page 113.

9.5 Installation Instructions

For your convenience, the programs described above are made available to you in the sample program set for this redbook (see 2.7.1.5, "Download and Install the SG245148 Package" on page 19). They are grouped in a CMS PACKAGE file. For easy handling issue:

```
FILEList JNRCMS PACKAGE (Filelist
```

To see the file description comments, issue:

```
Xedit JNRCMS PACKAGE
```

Except for the sources of the C programs, all files must be installed in a BFS.

In order to install the JNRCMS package into your BFS, you need access to MAINT′s 193 (for the LOADBFS command), admin authority for the default BFS filepool, VMSYS, and POSIX superuser privileges. Since the JNRCMS LOADBFS changes your /etc/profile, you may want to create a backup copy of it.

In order to install the JRNCMS package, simply enter the following command:

```
loadbfs jnrcms
```

The preceding command will:

1. create a new BFS filespace called _JNRCMS, with 1000 blocks
2. load all the executable and source file to that BFS filespace
3. create a symbolic link to it, named /usr/jnrcms
4. modify the CLASSPATH environment variable in /etc/profile to include /usr/jnrcms

Note: Java requires the path to C programs to be fully qualified. Therefore, the programs "jcopipe" and "jnrcms" must be installed in the indicated directories unless the CMSRexx.nrx and CMSPipe.nrx programs are changed and recompiled.

Testing

There are six sample programs that you can run and use as templates to get you started.

To test the installation of CMSRexx run:

```
java testCMSRexx
```

To test the installation of CMSPipe run:

```
java testCopipeReadto1 cp q time
```

Loading the JNRCMS package on a different filepool

If you want to install the JNRCMS package in a different file pool, create a copy of the JNRCMS LOADBFS file and modify the VMSYS string to the filepool name that you want.

Chapter 10. The GUIMON Sample Program

GUIMON is the major sample application provided with the book. It uses quite a few components. Three parts can be distinguished in GUIMON:

The client A NetRexx program using a GUI to show VM performance data. This program must run on a system supporting Java's GUI class, AWT (for example, OS/2, IBM Network Station, or Windows/NT).

This part of GUIMON is called ***the client*** in the remainder of this chapter.

The server A NetRexx server program running in VM/ESA. It waits on a TCP/IP port for requests from a GUIMON client.

This part of GUIMON is called ***the server***.

The monitor A VM service machine collecting performance data and/or a set of CMS files containing historical performance data.
This part of GUIMON is called ***the monitor***; the files created or held by the monitor are named ***PERFLOG*** files.

With the GUIMON sample program, we want to illustrate a few facts:

- Creating networked applications with Java and NetRexx is easy.
- Existing data in VM can be presented in a modern way.
- Exploiting the multitasking provided with Java and NetRexx is simple.

In the remainder of the chapter, a detailed study of the involved code is not attempted. Concentration is at a higher level: what are the components, and how do they interact. This should give you enough information to allow you to enhance GUIMON, or use GUIMON to display other kinds of data.

10.1 GUIMON - Pictures

The first picture of GUIMON, shown in Figure 27, prompts the user to enter the address of a VM host.

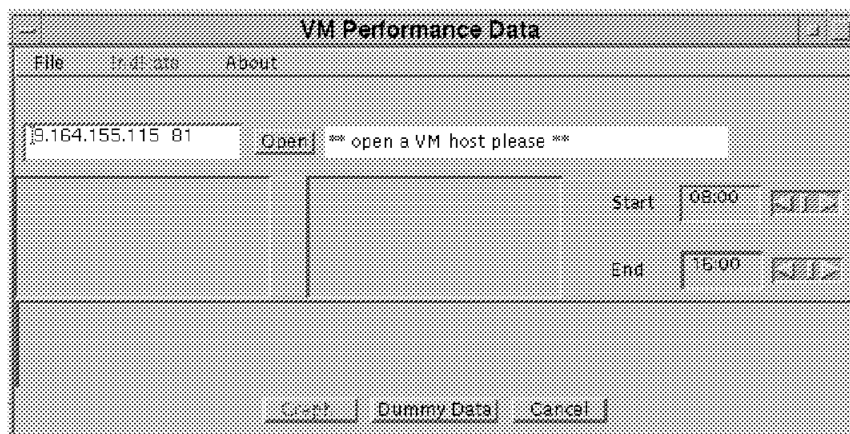


Figure 27. GUIMON Prompting for Host Address

© Copyright IBM Corp. 1998

After entering an IP address and a port number, the end-user must select the **Open** push button. GUIMON then seeks contact with GUIMON′s server in the indicated host.

If all goes well, the frame should look like Figure 28.

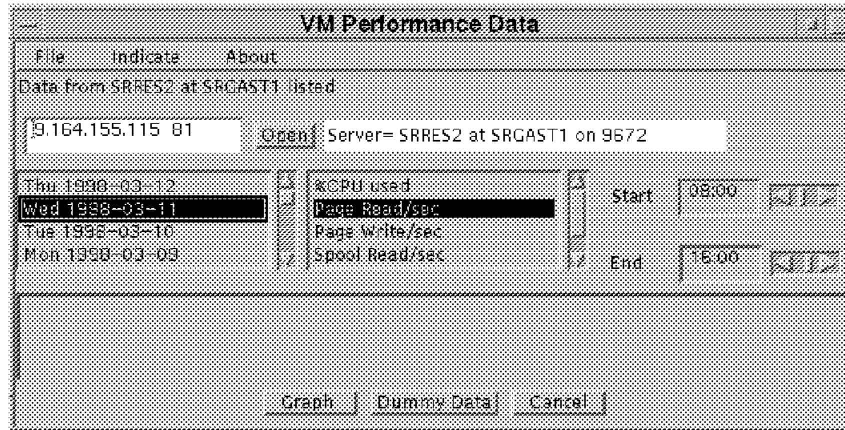


Figure 28. GUIMON Listing the Performance Information Found

It can be seen that the server sent back the performance information that is available: which files do exist and which performance indicators can be plotted.

In the picture, two list boxes can be seen. The one on the far left displays the dates for which performance files are available. The one in the middle lists the performance values found in these files. Using the scroll bars on the right-hand side of these list boxes, the end-user can locate the time period and statistic to plot.

When the **Graph** push button is selected, a graphic is shown, such as the one that can be admired in Figure 29.

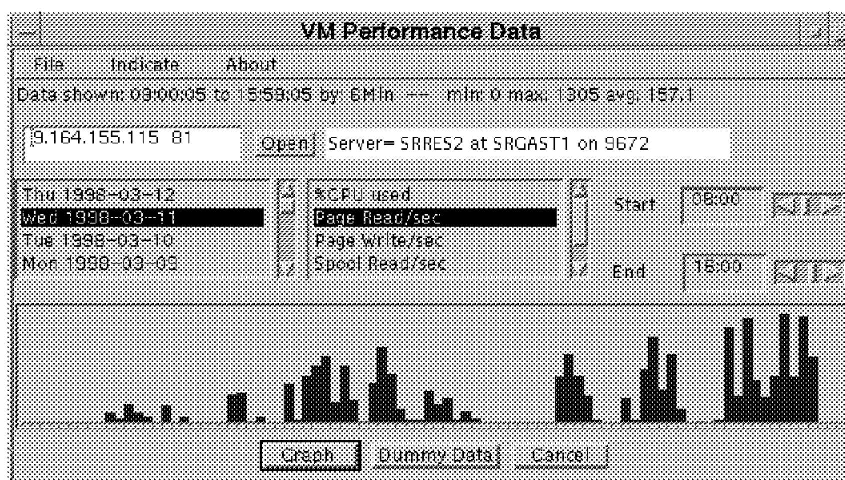


Figure 29. GUIMON Plotting a Performance Variable

Selecting the Graph push button is not the only way to get a plot; a double click in the files list box or in the performance variables list box is as effective.

GUIMON has one other feature: the **Indicate** pull down menu item can be used to get some real-time data from the VM server. The end-user can request the current usage of CPU, Minidisk cache, Storage, and Expanded storage. When an item is selected, the server executes a CP INDICATE command and sends the requested response back. As INDICATE keeps no historical information, the output is a line of text, shown at the top of the frame, not a graph.

Eye Pleasing?

It probably is clear to the reader that our time was not invested in producing the best possible look for the frame. Some more time should be invested to make the layout of GUIMON's frame better.

Having seen what GUIMON can produce, it is now time to explain how the GUIMON client/server program is installed. Later in this chapter an overview of the GUIMON program logic is given, including the communication protocol between the client and the server.

10.2 GUIMON - Installation Instructions

The installation instructions are grouped in three sections: the monitor, the server, and the client. All of the files required to install the entire GUIMON application are included with the sample program set for the redbook (see 2.7.1.5, "Download and Install the SG245148 Package" on page 19).

10.2.1 Installing the GUIMON Monitor

The files composing the GUIMON monitor are all listed in MONREAD PACKAGE. For easy handling issue the following command:

```
FILEList MONREAD PACKAGE (Filelist
```

To see the file description comments, issue:

```
Xedit MONREAD PACKAGE
```

To install the GUIMON monitor, there are two alternatives:

- Just install our sample PERFLOG files, from which performance data of our system can be shown.
- Install the MONREAL EXEC, and thus capture real performance data of your system. Note that GUIMON's monitor can perfectly well coexist with (for example) MONWRITE since more than one user can be connected to CP's monitor service.

10.2.1.1 A Dummy Installation

For a dummy installation, just copy the following files to some minidisk or SFS directory:

```
GETDATA  REXX        ** a Pipe subroutine used by the server
PERFLOG  DESCRIBE    ** the file describing the PERFLOG files
19980310 PERFLOG     ** sample files with performance data
19980309 PERFLOG
19980306 PERFLOG
19980305 PERFLOG
```

10.2.1.2 A Real Installation

For a real monitor installation, proceed as follows.

1. Decide what the monitor gets as A-disk: a minidisk or an SFS directory. We recommend an SFS directory as this allows GUIMON's server to have access to the latest collected data. If a minidisk is being used, the SERVER should issue an ACCESS command before LISTFILE and GETDATA requests are handled. The coding in `ServerHandler.nrx` does **not** execute such an ACCESS command.

   If using an SFS directory, be sure to XAUTOLOG GUIMON's monitor only after the SFS server has been started. To enroll the monitor in the SFS, issue

   ```
   ENROLL USER monread VMSYSU (BLOCKS 2000
   ```

2. Create a GUIMON monitor user ID. The sample directory entry we provide, as MONREAD DIRECT, looks as follows:

   ```
   USER MONREAD MONREAL  16M 64M EG
    MACHINE XA
    SPOOL 000C 2540 READER *
    SPOOL 000D 2540 PUNCH A
    SPOOL 000E 1403 A
    CONSOLE 009 3215 T
    LINK MAINT 0190 0190 RR
    LINK MAINT 019D 019D RR
    LINK MAINT 019E 019E RR
    IUCV *MONITOR PRIORITY MSGLIMIT 255
    IPL CMS PARM FILEPOOL VMSYSU AUTOCR
   ```

3. Copy the following files to the monitor's 191 minidisk or SFS directory:

   ```
   GETDATA  REXX      ** a Pipe subroutine used by the server
   PERFLOG  DESCRIBE  ** the file describing the PERFLOG files
   MONREAD  PROFILE   ** sample PROFILE EXEC
   MONREAD  EXEC      ** the real code
   ```

4. Have a look in MONREAD EXEC; a few variables can be tailored:

   ```
   /*--------------------- Tailoring -------------------------------*/
   MonitorSegment='MONDCSS'        /* Name of monitor NSS */
   ErrorsToUserid='SRRES2'         /* Userid can be RSCS */
   ErrorsToTagInfo=''              /* A must if RSCS; else must be blank */
   KeepLogs=14                     /* How many performance logs to keep  */

   /*--------------------- End -------------------------------------*/
   ```

5. XAUTOLOG the monitor user.

Verify the operation:

- Directly when started, it should at least create a *yyyymmdd PERFLOG* file on its A-disk/directory.
- Verify that the CP monitor is started (the MONREAD EXEC tries to start it if required) and that your GUIMON's monitor user is connected to it.

```
⋮
MONITOR SAMPLE ACTIVE
               INTERVAL    1 MINUTES
               RATE     2.00 SECONDS
MONITOR DCSS NAME - MONDCSS
CONFIGURATION SIZE      241 LIMIT         1 MINUTES
CONFIGURATION AREA IS FREE
USERS CONNECTED TO *MONITOR - MONREAD
MONITOR   DOMAIN ENABLED
SYSTEM    DOMAIN ENABLED
PROCESSOR DOMAIN DISABLED
STORAGE   DOMAIN DISABLED
USER      DOMAIN ENABLED
⋮
```

Note: The MONREAD EXEC has difficulty knowing when the data for a monitor interval is complete. Therefore, the data is written with a delay of one monitor interval. An example, supposing a monitor interval of three minutes, could be this:

```
23:09:00 1 record with CPU consumption
           between 23:06:00 and 23:09:00
           for CPU 1 arrives
23:09:00 2 record with CPU consumption
           between 23:06:00 and 23:09:00
           for CPU 2 arrives
⋮
23:12:00 3 record with CPU consumption
           between 23:09:00 and 23:12:00
           for CPU 1 arrives
```

At **3**, MONREAD detects a "long" delay between the records, and so decides a new monitor interval has started. So only at 23:12:00 can it calculate the average CPU consumption between 23:06:00 and 23:09:00 by calculating the average of records **1** and **2**. Similarly, only at 23:15:00 is MONREAD able to calculate the average CPU consumption for the interval 23:09:00 to 23:12:00.

The data is correctly calculated (we hope), but it appears in the PERFLOG file a bit late. Avoiding this delay seems possible by making the Pipeline in the MONREAD EXEC more complex by inserting some extra DELAY stages.

10.2.2 Installing the GUIMON Client

The client of GUIMON can be any workstation with Java and NetRexx support. NetRexx installation instructions for IBM Network Station users are provided in 11.8, "Setting up to Run Java and NetRexx Programs" on page 128. To install NetRexx on another platform, refer to:

`http://www2.hursley.ibm.com/netrexx/`

When the workstation setup for Java and NetRexx is OK, make sure that the workstation has access to the files listed below. Two alternatives seem to be available:

- download them from VM to the workstation, or
- leave them in the BFS on VM and use VM's new NFS support in TCP/IP to mount the BFS directory on the workstation.

In any case, the workstation must have access to the following class files, in a directory that is included in the workstation's CLASSPATH.

```
--- classes compiled out of GuiMon.nrx --------------------------------
 GuiMon.class                the base program
 GuiMonActions.class         action handling
 GuiMonFrame.class           the frame it creates
 GuiMonFrameController.class closes frame
 GuiMonMenuAction.class      handles menu actions
 GuiMonTimeScroll.class      handles movement of the two scroll bars
--- class compiled out of GetPerf.nrx --------------------------------
 GetPerf.class               forward requests from GuiMonXxxx to VM
--- classes compiled out of AboutFrame.nrx ---------------------------
 AboutFrame.class            the base program
 AboutActionClass.class      action handling
 AboutFrameController.class  closes frame
--- class compiled out of TimeGrap.java ------------------------------
 TimeGraph.class             produces the bar-chart like graph
```

The file identifiers are too long to fit in CMS's normal filename filetype structure, so they are delivered with a temporary CMS name. Refer to 10.2.4, "Installing Client and Server Files" on page 107 for more information.

10.2.3 Installing the GUIMON Server

The server uses a few NetRexx programs and one C program (the C program is the interface with CMS Pipelines). As the GUIMON server runs under VM, the class files and the C program must be installed in the BFS.

Here is the list of the class files and the C program.

```
  TCPServer.class              provides multi tasking support
  MyServer.class               starts a TCPServer object
  ServerHandler.class          the core code of the GUIMON server
 --- parts of JNRCMS -------------------------------------------
  CMSPipe.class                interface with C program
  jcopipe                      C program taking to CMS Pipelines
```

The .class files must be installed in a BFS directory included in the server's CLASSPATH. Note that GUIMON's server uses the "JNRCMS" code, explained in 9.4, "Running CMS Pipelines with NetRexx" on page 96, so that must be installed too. The C program must be installed in a hardcoded BFS directory (see 9.5, "Installation Instructions" on page 100).

The server can run in any CMS user, on two conditions:

- The minidisk (or SFS directory) of the GUIMON monitor must be accessed as P. If SFS is used, the following GRANT commands should be given:

  ```
  GRANT AUTH mypool:myMonitor. TO myServer (READ NEWREAD
  GRANT AUTH * * mypool:myMonitor. TO myServer  (READ
  ```

  If our names are used, these would be:

  ```
  GRANT AUTH VMSYSU:MONREAD. TO GUIMONSR (READ NEWREAD
  GRANT AUTH * * VMSYSU:MONREAD. TO GUIMONSR (READ
  ```

- The user must have BFS read authorization on the .class files and execute authorization on the C program.
- The user has mounted a filesystem, either by using OPENVM MOUNT, or as specified the CP directory (see "Automatic Mount" on page 23).
- The user's CLASSPATH must include the BFS directories where JNRCMS and GUIMON are installed. The LOADBFS file for GUIMON and JNRCMS will update the CLASSPATH in /etc/profile. This means that the CLASSPATH is

OK for POSIX Shell users. CMS users must update their CLASSPATH definition in GLOBALV, for example by executing SETCENV GETSHELL CLASSPATH.

This is what we use to start the server:

```
'ACCESS VMSYSU:MONREAD. P'
if rc<>0 then do
   Say 'Can not run. Disk P must be accessed.'
   ...  cry for help  ...
   exit 9876
end
'EXEC SETCENV GETSHELL CLASSPATH' /* get CLASSPATH from /etc/profile*/
'EXEC NRR /usr/guimon/server/MyServer.class 81'
```

Note that we start the server using the NRR EXEC since it takes care of GLOBALling the C runtime library and setting the current directory to the path of the class file. This means that the server must have access to the place where the NRR EXEC is installed. Alternatively, insert the following in the server's startup EXEC:

```
'ACCESS VMSYSU:MONREAD. P'
if rc<>0 then do
   Say 'Can not run. Disk P must be accessed.'
   ...  cry for help  ...
   exit 9876
end
'GLOBAL LOADLIB SCEERUN'
'EXEC SETCENV GETSHELL CLASSPATH' /* get CLASSPATH from /etc/profile*/
'EXEC OPENVM SET DIR /usr/guimon/server/'
'EXEC OPENVM RUN java MyServer 81'
```

Note: The SETCENV EXEC, explained in 3.8, "SETCENV - Setting C Environment Variables" on page 33, is also included in the sample program set for the redbook (see 2.7.1.5, "Download and Install the SG245148 Package" on page 19).

10.2.4 Installing Client and Server Files

First of all it is important to remember that the server uses JNRCMS, the Java - CMS Pipelines interface provided with this book (see 9.4, "Running CMS Pipelines with NetRexx" on page 96). Therefore, JNRCMS must be installed before one can use or compile the GUIMON NetRexx programs (refer to 9.5, "Installation Instructions" on page 100).

To install GUIMON's client and server programs into a BFS, a LOADBFS control file is provided. A LOADBFS file contains instructions to create BFS objects (such as files and directories). LOADBFS files are handled by the LOADBFS EXEC, which can be found on MAINT 193.

You may like to tailor "GUIMON LOADBFS" to your needs. An explanation of the LOADBFS control instructions can be found in the LOADBFS EXEC.

Basically the GUIMON LOADBFS file defines where the GUIMON files will be installed. There are three types of files: for the server, for the client, and the source files. You may or may not want to install all three types in the same location. The sample LOADBFS file:

1. creates a BFS file space named _GUIMON,
2. stores the source and the class files for the server in the "/guimon/server" directory,

3. stores the source and the class files for the client in the "/guimon/client" directory, and
4. creates "/usr/guimon/" as symbolic link, so you can easily find the GUIMON application in "/usr."

In order to install the GUIMON package into your system, you need access to MAINT′s 193 (for the LOADBFS command), admin authority for the default BFS filepool, VMSYS, and POSIS superuser authority. Since the GUIMON LOADBFS changes your /etc/profile, you may want to create a backup copy of it.

In order to install GUIMON, type the following command:

```
loadbfs guimon (noisy
```

It is wise to verify the installation a bit, especially the CLASSPATH. Issue the following commands:

```
openvm mount /../VMBFS:VMSYS:ROOT/ / -- Mount a file system
bfslist /usr/guimon/server           -- 5 files are expected
bfslist /usr/guimon/client           -- 15 files are expected
xedit /etc/profile (namet bfs        -- Verify CLASSPATH
```

The classpath is expected to be a long line, as:

```
export CLASSPATH=$CLASSPATH/usr/jav/usr/java/lib/classes.zip.......
```

We expect the following directories to be included:

```
/usr/java/classes    and   /usr/java/lib/classes.zip
/usr/NetRexx/bin     and   /usr/NetRexx/lib/NetRexxC.zip
/usr/guimon/server   and   /usr/jnrcms
```

When the classpath looks fine, execute `SETCENV GETSHELL CLASSPATH` in all users that will run the GUIMON server code (or that plan to use JNRCMS).

Using the GUIMON Client on a Workstation

When, for example, the client code has to be installed on a PC, but you did not install the files into the BFS, you need to know the relation between the CMS fileid and the real name. Here is a table, extracted from the GUIMON LOADBFS file, showing the equivalences:

CMS fname ftype	type	Install as
TCPSER1 CLASSBIN	server	TCPServer.class
TCPSER2 NRX	source	TCPServer.nrx
MYSERVER NRX	source	MyServer.nrx
MYSERVER CLASSBIN	server	MyServer.class
SERVER1 CLASSBIN	server	ServerHandler.class
SERVER2 NRX	source	ServerHandler.nrx
CMSPIPE CLASSBIN	server	CMSPipe.class
CMSPIPE NRX	source	CMSPipe.nrx
JCOPIPE NONEBIN	server	jcopipe
TIMEGR1 JAVA	source	TimeGraph.java
TIMEGR2 CLASSBIN	client	TimeGraph.class
GUIMON NRX	source	GuiMon.nrx
GUIMON CLASSBIN	client	GuiMon.class
GUIMON1 CLASSBIN	client	GuiMonActions.class
GUIMON2 CLASSBIN	client	GuiMonFrame.class
GUIMON3 CLASSBIN	client	GuiMonFrameController.class

GUIMON4	CLASSBIN	client	GuiMonMenuAction.class
GUIMON5	CLASSBIN	client	GuiMonTimeScroll.class
GETPERF	NRX	source	GetPerf.nrx
GETPERF	CLASSBIN	client	GetPerf.class
ABOUTA1	CLASSBIN	client	AboutActionClass.class
ABOUTF1	CLASSBIN	client	AboutFrame.class
ABOUTF2	CLASSBIN	client	AboutFrameController.class
ABOUTF3	NRX	source	AboutFrame.nrx

When the CMS filetype ends in BIN, the file must be transferred as a binary file. The “type” column indicates who needs the file, the client or the server; files of type “source” are required only if you want to modify or study the GUIMON application.

One of the possible methods to transfer files from VM to workstations is FTP. Here is an example of how FTP can be used on a PC to obtain files stored in an SFS on VM.

```
C:\>ftp 9.164.xxx.yy
Connected to 9.164.xxx.yy.
220-FTPSERVE IBM VM V2R4 at SRGAST1.YOUR.DOMAIN.NAME, 13:56:32 CET WEDNESDAY 03
/11/98
220 Connection will close if idle for more than 5 minutes.
User (9.164.xxx.yy:(none)): srres2
331 Send password please.
Password:
230 SRRES2 logged in; no working directory defined
ftp> cd vmsysu:srres2.
250 SFS working directory is VMSYSU:SRRES2.
ftp> bin
200 Representation type is IMAGE.
ftp> get nnnnn.CLASSBIN
200 Port request OK.
150 Storing file 'nnnnn.CLASSBIN'
250 Transfer completed successfully.
31574 bytes sent in 0,12 seconds (263,12 Kbytes/sec)
```

In the next example FTP is used to directly extract the files from the VM BFS.

```
C:\>ftp
ftp> open 9.164.yyy.xxx
Connected to 9.164.yyy.xxx.
220-FTPSERVE IBM VM V2R4 at SRGAST1.YOUR.DOMAIN.NAME, 14:21:57 CET WEDNESDAY 03
/11/98
220 Connection will close if idle for more than 5 minutes.
User (9.164.yyy.xxx:(none)): srres2
331 Send password please.
Password:
230 SRRES2 logged in; no working directory defined
ftp> cd /../VMBFS:VMSYS:ROOT/
250 BFS working directory is /../VMBFS:VMSYS:ROOT/
ftp> cd home/kris
250 BFS working directory is /../VMBFS:VMSYS:ROOT/home/kris/
ftp> lcd /temp
Local directory now C:\TEMP
ftp> bin
200 Representation type is IMAGE.
ftp> get GuiMon.class
200 Port request OK.
150 Sending file 'GuiMon.class'
250 Transfer completed successfully.
994 bytes received in 0,06 seconds (16,57 Kbytes/sec)
```

Note: To use BFS in FTP, the cd command must be done in two steps as shown above; the next connection request in one step fails:

```
cd /../VMBFS:VMSYS:ROOT/home/kris
```

10.3 GUIMON - Functional Overview

In this section an overview of the GUIMON application is provided. As mentioned before, GUIMON is composed of three parts: a client running on a workstation, a server running on VM, and a performance monitor also running on VM. First, the easiest part is discussed: the monitor.

10.3.1 GUIMON - the Monitor

This part of GUIMON is in fact optional for a GUIMON demonstration. What GUIMON needs is performance data in CMS files, called PERFLOG files, stored in a minidisk or SFS directory. The only requirement is that GUIMON's server can read them. The required record format is detailed in 10.5, “GUIMON Record Format Requirements” on page 117.

But, as we wanted to provide a complete demonstration package, a simple real-time performance monitor has been written, too. It is written using classic REXX and CMS Pipelines. The provided sample monitor extracts CPU usage, page read, page write, spool read, and spool write statistics. It could be extended easily to collect more data.

There is only one important program, MONREAD EXEC. It does all that is required:

- If the Monitor saved segment does not exist, it creates it.
- If the CP monitor is not started, it will be started.
- A CMS Pipeline is started:
  - It connects to the monitor and extracts the record types required to calculate CPU usage, page read, pages write, spool read, and spool

write statistics. The data is written to a normal CMS file named *yyyymmdd PERFLOG A*. An extract from our test system looks like:

```
* Time     %CPU     PageR   PageW  SpoolR  SpoolW
21:30:01    0.17        0       0       0       0
21:31:01    9.10      183    5741       0       0
21:32:01   38.68      884     700       0       0
21:33:01   37.16      191       0       2       4
```

– Just before midnight, the connection to the monitor is stopped. After midnight, PERFLOG files that are too old are erased and a new one is started. (A variable in the MONREAD EXEC defines how many PERFLOG files are kept.)

10.3.2 GUIMON - the Server

The major task of the GUIMON server is twofold: list the performance files created by the monitor, and read such a file so that the client can produce a graph.

The GUIMON server is a NetRexx application that waits on a TCP/IP port for incoming requests. The requests generated by the client are listed in 10.4, “GUIMON - the Client-Server Communication” on page 113.

The classes used are:

MyServer This is the starting point, the class that must be started by the Java virtual machine. It takes one argument, a port number (the default is 81). It calls only TCPServer, which will listen on the specified port.

TCPServer This class, explained in 8.3, “Simple TCP/IP Server” on page 88, allows the GUIMON server to handle multiple clients concurrently. When a request comes in on the designated port, TCPServer calls method “runUser” in class “ServerHandler.”

ServerHandler This is the main class to be written by the GUIMON programmer. It reads lines from the TCP/IP socket, each of which starts with a request keyword (explained in 10.4, “GUIMON - the Client-Server Communication” on page 113). It performs the requested action and sends the result (or ERROR information) back to the client.

All requests are handled by calling a CMS Pipeline. We can call CMS Pipelines by using the CMSPipe class.

CMSPipe This is the class that, using the “jcopipe” C program, makes CMS Pipelines available to Java and NetRexx programs. See 9.4, “Running CMS Pipelines with NetRexx” on page 96 for information.

In addition to these classes, the server needs:

jcopipe

The C program to interface with CMS Pipelines. It must be stored in a directory included in the PATH of the server.

GETDATA REXX

This is a Pipeline subroutine performing two functions that the server will request:

- obtaining the weekday for each performance file (they are named *yyyymmdd* PERFLOG).

- calculating averages: GUIMON′s graphic has been defined with room to display 90 numbers. When, for example, a whole day must be represented, averages must be calculated.

This means that GETDATA REXX must be stored on a minidisk or SFS directory which is ACCESSed by the server. To keep it simple, the installation instructions propose placing it on the GUIMON performance monitor′s A-disk.

This ends the overview of the GUIMON server.

10.3.3 GUIMON - the Client

The GUIMON client is a NetRexx application that presents a GUI to the end-user. End-user actions result in requests sent to the server. The requests are detailed in 10.4, "GUIMON - the Client-Server Communication" on page 113.

Three NetRexx source files are being used to create the GUIMON client: `GuiMon.nrx`, `GetPerf.nrx`, and `AboutFrame.nrx`.

After compilation more class files are created, a short description of each of which follows:

GuiMon
This is the starting point, the class that must be started by the Java virtual machine on the end-user′s workstation. The GUIMON source file defines more than a single class: all classes, listed below, whose name starts with GuiMon.

GuiMonFrame
The Frame of GUIMON is constructed by this class. It also includes supporting methods, such as one to show error messages. Two other methods are very important:

1. OpenVMSession() is called when "opening" a session with a VM system. OpenVmSession calls the GetPerf class; the information returned by GetPerf is then listed in the two list boxes on the frame.
2. GetTheData() is called when the end-user requests a graph. Also in this case, GetPerf is called and the information returned by GetPerf is then shown as a graph.

GuiMonActions
All actions caused by push buttons and listboxes are handled by methods defined here.

GuiMonFrameController
This class has only one function: end the application when the user selects "close" from the system menu.

GuiMonMenuAction
All pull down menu events are handled here. It is here that GUIMON′s AboutFrame can be called.

GuiMonTimeScroll
This class implements the adjustmentValueChanged() method which is called when the end-user changes the scroll bar. The two scroll bar objects define the start and end time of the data to plot. When the scroll bar′s position is changed, a time value is calculated out of it and displayed on the frame. In addition, we refuse any scroll bar

action that would cause the start time being bigger than the end time, or vice versa.

GetPerf
Methods in this class handle the communication with the VM host. The communication is explained in 10.4, "GUIMON - the Client-Server Communication."

AboutFrameXxxx
These classes are called when the user selects the "About" pull down menu entry. AboutFrame is completely explained in Chapter 5, "AboutFrame, a Reusable Class" on page 55.

10.4 GUIMON - the Client-Server Communication

As VM/ESA does not support Java's GUI (the AWT class), any VM application with GUI written in Java or NetRexx must be designed as a client-server application. This seems more difficult than it really is.

The usage of TCP/IP sockets from inside NetRexx (or Java) is very simple indeed. There are only two problems to be resolved:

- On the basic socket interface, one reads a number of bytes, and as long as that many bytes did not arrive the application waits. This problem is somewhat hidden by the Java methods we call: we read and write records. But if by accident the application is "talking" to a partner using another convention, you can end up waiting forever.
- The partners need to understand what they request from each other. For example, they must know when the data sent by the other side is complete to stop reading. Or in our case, we must know when the last record arrives and then stop reading.

The protocol used in GUIMON is kept very simple.

Client The client always sends a single record. A blank delimited keyword describes the request, and some requests also have blank delimited options.

Server The server replies to each request with one or more records. Each record starts with a keyword, too. The last record sent to the client always has a keyword DONE or ERROR.

To make it easier to study what GUIMON does, the application shows on the server console what records are sent or received.

10.4.1 Request Formats

To describe the record formats, we position ourselves at the client side. The requests are generated by the GUIMON client application, but any application using the protocol defined here could use GUIMON's server.

Following each request, the expected reply records are listed.

10.4.1.1 LISTFILE Request

This is the first request a client is expected to send; in response, the server tells which performance files are found and which data they contain. The request is generated when the end-user selects the "Open" push button.

Request record format:

```
LISTFILE
```

The server issues a few CMS Pipe commands and sends the records described below back. The code can be found in file *Serverhandler.nrx*, method *ListFile()*, an extract is reproduced in Figure 30 on page 115. The records sent back are:

IDENT To personalize the GUIMON frame a bit, an IDENT record can be sent back.

```
IDENT server at nodeid blabla ...
```

Only one IDENT record should be sent. The GUIMON client expects that the first three words read "userid at nodeid."

FILE FILE records specify the PERFLOG files found on the server's P disk. The records sent to the client look like:

```
FILE xxxYyyyMmDd xxxYyyyMmDd ...
```

For example:

```
FILE Thu19980312 Wed19980311 Tue19980310 Mon19980309
```

More than one FILE record is allowed; entries of all FILE records are appended by the client, forming one long list. Sending many entries per FILE record improves performance.

The GUIMON client reformats the entries a bit and lists them as:

```
Thu 1998-03-12
Wed 1998-03-11
Tue 1998-03-10
Mon 1998-03-09
```

DESC The DESC records define what the user can select as performance variables to plot. Each DESC record defines one entry, in the following format:

```
DESC performance_variable
```

Examples:

```
DESC %CPU used
DESC Page Read/sec
⋮
```

In practice, the VM server reads the file PERFLOG DESCRIBE, which is described in 10.5.1.2, "The PERFLOG DESCRIBE File" on page 118, and sends the information found therein to the client.

```
Method ListFile(input=BufferedReader,output=PrintWriter) -
              Signals InterruptedException
 /* -- Issue a PIPE to find who and where we are -----*/
   mypipe = CMSPipe('Command IDENTIFY'  - -- Issue an IDENTIFY cmd
                   '|Split|Take 3'      - -- Take first 3 words
                   '|Append CP Q CPUID' - -- What CPU is being used ?
                    '||Spec W3'         - --  Keep Cpuid only
                    '||Spec /on/ 1 9.4 Nw' - -- Only CPU Model
                   '|Join * / /'        - -- reconstruct 1 record
                   '|Change w2 /AT/at/' - -- Mixed case is nicer
                   '|fitting *>java')     -- output back to Java
   rc=mypipe.getPipeRc()  -- if PIPE already ended, syntax error
   if rc <> 0 Then do
      output.printLn('ERROR Syntax error' rc 'in IDENTIFY Pipe')
      output.flush();
      return
   end
   line = mypipe.readto()      -- read 1 PIPE record
   if line <> null then do
      output.printLn('IDENT' line)
      output.flush();
   end
   mypipe.endPipe()       -- Close this pipe nicely
```

Figure 30. Extract of GUIMON's Server Code. It illustrates how the DESC record is built and sent.

10.4.1.2 GETDATA Request

With the GETDATA verb the client requests the server to send the specified performance data back. The server extracts the specified time period out of the PERFLOG file, calculates averages and sends the result back. A GETDATA request is generated when the end-user selects the “Graph” push button.

Request record format:

```
GETDATA perf_file startTime endTime bySecs perf_var
```

Where:

```
"perf_file"  formatted yyyymddd, is the filename of a PERFLOG file
"startTime"  formatted hh:mm:ss, defines the start of the graph
"endTime"    formatted hh:mm:ss, defines the end of the graph
"bySecs"     is a number of seconds, used to calculate averages
"perf_var"   defines what to plot (it is a DESC record operand)
```

The server issues a CMS Pipe command to read the requested PERFLOG file and get the data. The code can be found in file *Serverhandler.nrx*, method *GetData()*; an extract is reproduced in Figure 31 on page 116. There is only one record-type that can be sent back:

```
DATA value1 value2 value3 ...
DATA valuen valuen+1 TIMES start end
```

As illustrated, multiple DATA records can be sent. Placing many values on a DATA record improves performance. A DATA record can contain a TIMES section which is used for the following reason:

The end-user might request a graph from 08:00:00 to 18:00:00. The PERFLOG file might, however, cover only the 09:30:00 to 15:12:00 time period. To inform the user, a TIMES section can be used; in our case it would read `TIMES 09:30:00 15:12:00`. The GUIMON client code displays this TIMES information to the user.

```
getWord = perfloc[perf_desc]
parse startTime h':'m':'s  ; FromSec=(h*3600+m*60+s).right(5,0)

CalcSeconds= -                   -- PIPE: Convert hh:mm:ss to seconds
  'h: 1.2 . m: 4.2 . s: 7.2 .'-  -- Take Hours, minutes &
       'Print h*3600+m*60+s'  -  -- Convert to seconds
       'Picture 99999 1'         -- Print them with leading 0's

ThePipe ='<' perf_file 'PERFLOG' fmode-   -- Read the file
         '|UNPACK'    -                   -- Unpack it (if required)
         '|NFIND *'||  -                  -- No comment records
         '|FROMTARGET PICK 1.8 >>= "'startTime'"'-
         '|NOT FROMTARGET PICK 1.8 >> "'endTime'"'-
         '|SPEC'-                         -- Rearrange record:
             CalcSeconds-                    -- Cvt hh:mm:ss to second
             getWord 'NextWord'-             -- Take nbr asked by user
         '|Rexx GETDATA' FromSec bySecs-  -- Calc average by time
         '|JOIN 20 / /'-                  -- Group per 20 items
         '|fitting *>java'                -- Pass to Java
mypipe = CMSPipe(ThePipe)                 -- Run the Pipeline
...  test returncode ...

 loop forever               -- reading PIPE records: REXX style ..
    line = mypipe.readto()   -- Read a record
    rc=mypipe.getReadtoRc()  -- Was there a record ?
    If rc<>0 then leave      --   no, leave loop
    output.printLn('DATA 'line)
 end
 mypipe.endPipe()            -- Close this pipe nicely
 output.printLn('DONE');  output.flush()
```

Figure 31. Extract of GUIMON's Server Code. It illustrates how the data is obtained from a PERFLOG file.

10.4.1.3 INDICATE Request

To make some use of pull down menu entries, we decided it might be nice if the end-user could request some real-time information. This information is extracted from CP's INDICATE command.

Thanks to our general client-server communication structure, adding these extra items was not difficult, and could be done with little effort.

INDICATE request record format:

```
INDICATE keyword
```

Where *keyword* basically is the string which identifies the beginning of the relevant part of INDICATE's response.

The server code, that can found in file *Serverhandler.nrx*, method *GetIndicate()*, is trivial; it is reproduced in Figure 32 on page 117.

```
Method GetIndicate(input=BufferedReader,output=PrintWriter,request=Rexx)-
              Signals  InterruptedException
  parse request . what .
  Select
   when what='CPU'  then find='AVGPROC'
   when what='MDC'  then find='MDC'
   otherwise            find=what
  end
 /* -- Issue a PIPE to get the current INDICATE -----*/
   mypipe = CMSPipe('CP INDICATE'       - -- Issue an INDICATE
                   '|Find'  find ||      - -- Take first 3 words
                   '|Join * / /'         - -- reconstruct 1 record
                   '|fitting *>java')     -- output back to Java
   rc=mypipe.getPipeRc()  -- if PIPE already ended, syntax error
   if rc <> 0 Then do
      output.printLn('ERROR Syntax error'  rc 'in INDICATE Pipe')
      output.flush();
      return
   end
   line = mypipe.readto()      -- reading 1 PIPE record
   mypipe.endPipe()       -- Close this pipe nicely
   if line = null then do
      if what='XSTORE'
        then line= 'Expanded storage is not available on this VM system.'
        else line= what 'is not found in CP INDICATE'
   end
   output.printLn('INDICATE'  line);output.flush()
   output.printLn('DONE');  output.flush()
```

Figure 32. Extract of GUIMON's Server Code - INDICATE Request

10.5 GUIMON Record Format Requirements

Without changing the GUIMON code, GUIMON can be easily used to allow your end-users to plot other performance data. Only the format of the PERFLOG files and the PERFLOG DESCRIBE file must be understood and adapted.

With minimal effort the GUIMON could be changed a bit and so allow the plot of other VM data.

10.5.1.1 The PERFLOG File Format

Each PERFLOG file must have a date as filename, formatted yyyymmdd; the filetype must be PERFLOG and it must reside on the P-disk of the GUIMON server.

For the record formats, there is only one restriction: each record must start with a timestamp formatted hh:mm:ss. The remainder of each record contains data; almost any format will do, so long as CMS Pipeline's SPECS stage can convert it to a number. The PERFLOG file made by GUIMON's performance monitor is simple:

```
* Time    %CPU     PageR   PageW  SpoolR  SpoolW
09:03:01   1.48       0       0       0       0
09:04:01   1.88       2       0       0       4
09:05:01  12.49     157     379       0       3
09:06:01   4.85       0       0       0       0
09:07:01   0.76       0       0       0       0
09:08:01  57.12     756    6525       0       0
09:09:01   9.49     574      71       0       0
```

Thanks to the PERFLOG DESCRIBE file, other performance files can be handled by GUIMON too. The only real requirement is that they contain a time flag in columns 1-8.

10.5.1.2 The PERFLOG DESCRIBE File

The PERFLOG DESCRIBE file describes what performance data is available in the PERFLOG files. The file describing our PERFLOG files looks as follows:

```
%CPU used = w2
Page Read/sec = w3
Page Write/sec = w4
Spool Read/sec = w5
Spool Write/sec = w6
```

The general format is:

```
performance variable = SPECS parms
```

Where:

performance variable is what the end-user will see as description.

= is a separator that must be preceded and followed by one blank.

SPECS parms defines how a CMS Pipelines SPECS stage can get the data.

As an example, the first line in our PERFLOG DESCRIBE file tells us that performance variable "%CPU used" is "word 2" of each record.

If you would have a file where "%CPU used by CP" is stored in columns 20 to 27 as a floating point number, and "%Minidisk cache hits" in columns 45 to 47, you would code:

```
%CPU used by CP = 20-27 c2f
%Minidisk cache hits = 45-47
```

When the server reads the PERFLOG file, it stores the relation between the performance variables and the SPECS operand to use in a NetRexx compound variable. By consulting the compound variable, the server knows which data to extract when the end-user sends a request.

Figure 33 on page 119 contains the NetRexx code used for the compound variable. In classic REXX, the same could be achieved with a stem variable.

```
 class ServerHandler public extends TCPServer
   Properties Inheritable
1      perfloc = Rexx  -- Indexed variable holding location of perf data
   :
 Method ListFile(input=BufferedReader,output=PrintWriter)
   :
  /* -- Read file describing the performance data files ------------*/
    mypipe = CMSPipe('< PERFLOG DESCRIBE' fmode    - -- read the file
                '|Nfind *|Strip Trailing'      - -- cleanup a bit
                '|fitting *>java')
    ... check retcode ...
2     perfloc = ''                -- Now actually create the "stem"
    loop forever              -- reading PIPE records: REXX style ..
       line = mypipe.readto()   -- Read
       ... check retcode ...
       rc=mypipe.getReadtoRc()  -- Was there a record ?
       parse line desc ' = ' loc .
3        perfloc[desc]=loc
       output.printLn('DESC 'desc)
       output.flush()
    end
   :

 ----------------------------------------------------------------------
 Method GetData(input=BufferedReader,output=PrintWriter,request=Rexx) -
   :
4     getWord = perfloc[perf_desc]
   :

   CalcSeconds= -                   -- PIPE: Convert hh:mm:ss to seconds
     'h: 1.2 . m: 4.2 . s: 7.2 .'-  -- Take Hours, minutes & seconds
          'Print h*3600+m*60+s'  -  -- Convert to seconds
          'Picture 99999 1'         -- Print them with leading 0's

   ThePipe ='<' perf_file 'PERFLOG' fmode-   -- Read the file
            '|UNPACK'    -                   -- Unpack it (if required)
            '|NFIND *'||  -                  -- No comment records
            '|FROMTARGET PICK 1.8 >>= "'startTime'"'-
            '|NOT FROMTARGET PICK 1.8 >> "'endTime'"'-
            '|SPEC'-                         -- Rearrange record:
                CalcSeconds-                    -- Cvt hh:mm:ss to second
                getWord 'NextWord'-             -- Take nbr asked by user
            '|Rexx GETDATA' FromSec bySecs-  -- Make average by time
            '|JOIN 20 / /'-                  -- Group per 20 items
            '|fitting *>java'                -- Pass to Java
   mypipe = CMSPipe(ThePipe)
```

Figure 33. Extract of GUIMON's Server Code - PERFLOG DESCRIBE Example. It illustrates how the DATA are obtained from a PERFLOG file.

At line **1** we define the compound variable as a class property. This is required as the compound variable is used by more than one method of the class (in simple words: it must be a kind of global variable).

2 Here we actually initilize the compound variable and give a default value to all tails. This compares to Rexx's statement `myStem.=''`

3 For each record in the PERFLOG DESCRIBE file we create a compound variable. Expanding on our sample PERFLOG DESCRIBE file, one entry would look like: `perfloc[%CPU used]=w2`

4 Here the compound variable is consulted.

Chapter 11. Running NetRexx and Java Applications on a Network Station

This chapter describes what you need to do before running your NetRexx and Java applications on an IBM Network Station. Complete information on installing and configuring IBM Network Stations may be found in the redbook *S/390 - IBM Network Station - Getting Started*, SG24-4954.

11.1 Network Computing - Extending VM/ESA Resources into the Network

With VM/ESA V2.3.0 acting as a server and with the Network Station family, you can take data and software off your desktop and put it on the network. Each member of the IBM Network Station family of network computers offers solutions to a wide range of business needs. Whether the demand is for basic server access to multiple servers, powerful Internet access, or Java application support, there′s a Network Station Series to suit your needs.

- Series 100: It′s about access
- Series 300: It′s about the Internet
- Series 1000: It′s about Java

This is the really good part.

The Network Station is a very flexible piece of equipment that gives customers, both large and small, access to:

- Terminal emulators for S/390, AS/400, and RS/6000 hosts
- Web browser for Internet and intranet applications and data
- Java virtual machine

  Run native Java applications. These applications execute in the Network Station, but reference data on one or several remote servers such as VM/ESA.

- IBM Network Station Manager software

  Hosts that you are able to access and the applications you will be using are determined by the network administrator. The administrator can configure the Network Station so that a minimum of time is spent on configuration details involving communications protocols, IP addresses, and host names. In addition, you will not have to worry about upgrading software on the Network Station. Upgrading software is much easier since all software resides on the servers and not on the desktops.

Benefits

- Low initial cost and lower total cost of ownership compared to PCs.
- Broader functionality and connectivity than non-programmable terminals.
- Centralized file storage, backup, and desktop management.

© Copyright IBM Corp. 1998

11.2 VM/ESA as a Network Station Server

As a server platform for the IBM Network Station, VM/ESA can act in several different roles:

- Boot server

  VM/ESA can provide the Network Station with network parameters and boot support. The two boot server protocols currently implemented are BOOTP (Boot Protocol) and TFTP (Trivial File Transfer Protocol).

- Management server

  VM/ESA can manage all Network Station configuration and preference files. This is done by the IBM Network Station Manager (NSM).

- Application server

  VM/ESA can serve a variety of applications to Network Stations; examples include:

  - IBM Network Station Browser (5648B08A)
  - Terminal emulators (tn3270 and tn5250)
  - User applications written in Java and NetRexx

11.3 Support Delivery Mechanism

Network Station Client support is not part of TCP/IP 310, nor is the Network Station Manager support part of VM/ESA V2.3.0.

This is because all the Network Station support will become a separate product. Until the product is available, there will be a 2.5+ level of the Network Station code available from the VM Network Station page:

```
http://www.vm.ibm.com/networkstation/
```

11.4 Hardware Requirements for VM/ESA

There are no special hardware requirements.

11.5 Software Requirements for VM/ESA

TCP/IP 310 is a priced feature of VM/ESA V2.3 that is installed with the operating system. TCP/IP V2.4 will be supported in VM/ESA V2.3 but does not exploit any of the new CP and CMS functions. For more information about TCP/IP see:

```
http://www.vm.ibm.com/related/tcpip/
```

The VM/ESA OpenEdition Shell and Utilities feature is required to install the IBM Network Station common code into the Byte File System (BFS).

To use the Network Station Manager, required for customizing Network Station user profiles, you will need both a JavaScript capable Web browser and a Web server. Three Web servers for VM/ESA are available. The following list briefly describes each of them. Further information may be found in the redbook *Web Server Solutions for VM/ESA*, SG24-4874.

- Webshare is a no-cost Web server written in CMS Pipelines and REXX. It gives VM the ability to serve text, graphics, sound and video on the Web.

- EnterpriseWeb from Beyond Software Inc. is built on the Webshare base. The original code was modified to use the CMS Pipelines dispatcher to interleave requests with I/O operations.
- VM:Webserver from Sterling Software is built on an integrated assembler nucleus using technology ported from existing Sterling applications.

11.6 Major Steps to Install VM/ESA Network Station Code

1. Download the Network Station code
2. Prepare for the installation
3. Plan the Byte File System file space structure
4. Install the Network Station code
5. Perform Network Station customization

11.6.1 Download the Network Station Code

See the VM Network Station home page at URL

```
http://www.vm.ibm.com/networkstation/
```

for information on how to obtain the Network Station code. There you will find step by step instructions for downloading and installing the Network Station code. In the following sections we provide a summary of the installation procedure based on our experience, highlighting those points that may deserve special consideration. You should, however, follow the instructions provided on the Web to ensure that you are using the most current information available.

11.6.2 Prepare for the Installation of the Network Station Client Code

For the P735FALQ user ID (installation user for TCP/IP) make sure that:

- Sufficient virtual storage is defined. At least 128M is needed to process the TARBIN file.
- It is defined as a POSIX super user. The following control statement should appear in its user directory entry:

  POSIXINFO UID 0 GID 0

- It is enrolled with administrator authority in the Shared File System (SFS). See *VM/ESA File Pool Planning, Administration, and Operation*, SC24-5751, for a detailed explanation of SFS administrator authority. In our case, the command used was:

  ENROLL ADMINISTRATOR P735FALQ VMSYSU:

- The OpenEdition Shell and Utilities feature for VM/ESA is installed.

11.6.2.1 TFTPD Virtual Machine Definition

For complete information on configuring the TFTPD virtual machine, see *TCP/IP Function Level 310 Planning and Customization*, SC24-5847. The following items should be considered:

- The size of the virtual machine depends upon the number of files to be cached in virtual storage. A 32M virtual machine is a good starting point.
- The performance of the TFTP daemon may depend on the scheduling share value set for the TFTPD virtual machine. You might also want to specify ***OPTION QUICKDSP*** for TFTPD.

Note: As with any changes of this type, you should consult your local performance expert for guidance.

11.6.2.2 VMNFS Virtual Machine Support and Definition

The Network Station requires a file system with stream oriented files and a directory tree structure in which to keep the configuration files. The Network File System server available with TCP/IP 310 (VMNFS) supports mounting of Shared File System (SFS) and Byte File System (BFS) directories. This enhancement means that a Network Station can now access files on a VM/ESA boot or target server system using either NFS or TFTP.

In the current (2.5 or 2.5+) release of the VM Network Station code, TFTP is the only supported way of downloading the kernel to the Network Station. During the residency, however, we experimented briefly with VMNFS and were able to download the kernel successfully with it.

To use NFS instead of TFTP to download the kernel you need Network Station client code release 2.5 or 2.5+ and TCP/IP 310 with VMNFS installed on your VM/ESA 2.3.0 system. To configure VMNFS, follow the instructions in *TCP/IP Function Level 310 Planning and Customization*, SC24-5847. Be sure to change the TAG.ANONYMOUS to Yes in the IBM DTCPARMS file.

Information on configuring the IBM Network Station to use NFS may be found in the redbook *S/390 - IBM Network Station - Getting Started*, SG24-4954. The NFS boot directory must be specified as a fully qualified BFS directory path. Since VMNFS is not able to resolve mount external links, the QIBM file space must be specified explicitly in the NFS boot directory parameter; in our configuration, the value of this parameter was:

```
/../VMBFS:VMSYSU:QIBM/ProdData/NetworkStation/
```

Note: Official support of NFS will be available in the next release of the Network Station client code.

11.6.2.3 BOOTPD Virtual Machine Definition

Complete information on configuring the BOOTPD virtual machine may also be found in *TCP/IP Function Level 310 Planning and Customization*, SC24-5847. The following items should be considered:

- The required size of the BOOTPD virtual machine depends upon the number of entries in the ETC BOOTPTAB server table file. A 32M virtual machine is a good starting point.
- The performance of the BOOTP daemon may be influenced by the scheduling share value set for the BOOTPD virtual machine. You might also want to specify ***OPTION QUICKDSP*** for BOOTPD.

  Note: As with any changes of this type, you should consult your local performance expert for guidance.

11.6.2.4 NSLD Virtual MACHINE Definitions

The IBM Network Station Login daemon (NSLD) is part of the Network Station Manager software. NSLD performs user authentication, provides data for user configuration, and responds to client requests for login information about a user ID on the system. See the VM Network Station home page at URL

```
http://www.vm.ibm.com/NetworkStation/
```

for information on how to obtain the Network Station Manager and Network Station Login Server (NSLD) code.

11.6.3 Plan the Byte File System File Space Structure

The TFTP daemon mounts a Byte File System file space from which to download the Network Station kernel and other files to the Network Station. To hold the Network Station support file tree, we used the VMSYSU: file pool, following the default installation procedure. Figure 34 illustrates the Byte File System structure used in our environment.

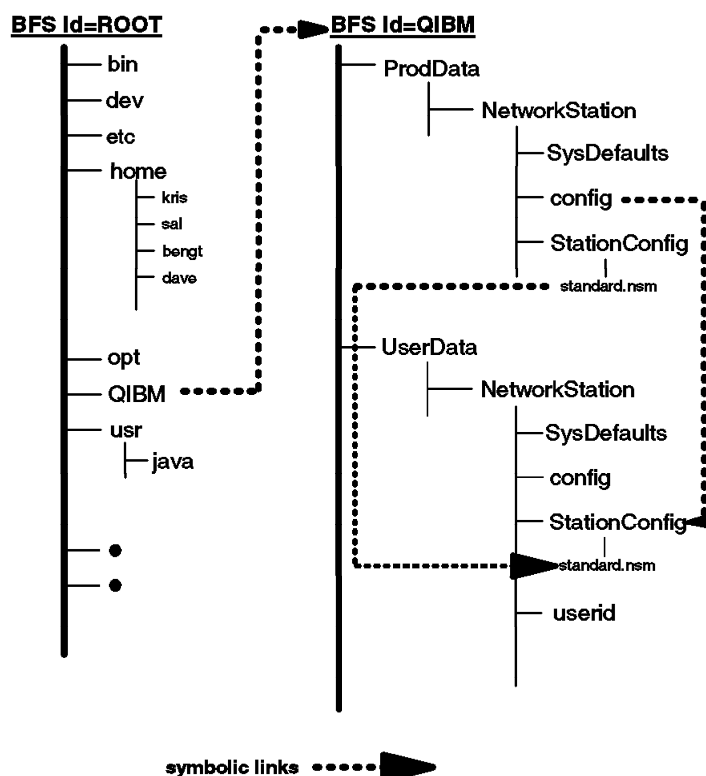


Figure 34. Byte File System Structure

Make sure that sufficient BFS file space is available to hold the client code files when the nets.tar file is exploded. The nets.tar file itself is approximately 7000 4K blocks in size, and an additional 10000 4K blocks are required to hold the expanded client code structure. Issue the following command to check that there is sufficient physical space available in VMSYSU:

QUERY FILEPOOL MINIDISK VMSYSU:

If you need to add more space, follow the instructions given in *CMS File Pool Planning, Administration, and Operation*, SC24-5751.

11.6.3.1 Build the BFS Structure for the IBM Network Station Support

If you plan to install the client code files into a byte file system other then VMSYSU, the NSTATION LOADBFS control file must be updated to specify the file space you wish to use. Also if you installed the Byte File System and ROOT directory in a filepool other than default (VMSYS), you need to update NSTATION LOADBFS references to VMSYS to reflect the BFS root file space on your system. The following steps create the byte file system that will contain the IBM Network Station client code:

- Log on to the installation user ID, P735FALQ, and access the minidisk that contains the NSTATION LOADBFS and NSTATION TARBIN files. Make sure that the MAINT 193 minidisk is accessed.
- Unpack the NSTATION TARBIN file using the command:

  ```
  COPYFILE NSTATION TARBIN fm = = = (OLDDATE REPLACE UNPACK
  ```

- Run the LOADBFS command against the NSTATION LOADBFS file. This will create the QIBM file space in the appropriate filepool. It will also create a mount external link (MEL) called /QIBM and write the NSTATION TARBIN file out to the BFS file space as nets.tar. The command to use is:

  ```
  LOADBFS NSTATION
  ```

  If there are any problems during this step, review the NSTATION LBFSLOG.
- Then mount the BFS root directory which contains the OpenEdition Shell and Utilities. **This command and others following are case sensitive.**

  ```
  openvm mount /../VMBFS:VMSYS:ROOT/ /
  ```

- Start the OpenEdition for VM/ESA shell and enter the shell environment.

  ```
  openvm shell
  ```

- If you are re-exploding the tar file into a structure you had created earlier, you must remove the earlier version of this structure.

  ```
  rm -Rf /QIBM/ProdData
  ```

- Change the current directory to /QIBM external link directory.

  ```
  cd /QIBM/
  ```

- Explode the tar file to create the necessary BFS directory structure and place all files in the appropriate directories.

  ```
  pax -rzf /nets.tar
  ```

- Remove the tar file from the Byte File System.

  ```
  rm /nets.tar
  ```

- Exit the OpenEdition for VM/ESA shell.

  ```
  exit
  ```

11.6.4 IBM Network Station Browser for VM/ESA

See the VM Network Station home page at URL

```
http://www.vm.ibm.com/NetworkStation/
```

for information on how to obtain the Network Station Browser. Follow the step by step instructions to download and install the Network Station Browser.

11.6.5 IBM Network Station Customization

11.6.5.1 IBM Network Station Configuration Files

The IBM Network Station Manager for VM/ESA should be used to configure your IBM Network Station networking environment. See the VM Network Station Web page

```
http://www.vm.ibm.com/NetworkStation/
```

for information on how to obtain the Network Station Manager and Network Station Login Server (NSLD) code.

We found that it was necessary initially to access the Network Station Manager from a PC-based Web browser in order to set options on the administrator′s ″setup tasks″ menu. After this had been done, the Network Station Manager could be accessed successfully from the Network Station Browser.

Also, before attempting to save any configuration information directly from applications such as the 3270 or 5250 emulators or the Network Station Browser, the user must first save preferences for each application using the Network Station Manager. The directory that these applications will try to write to does not exist until this is done.

11.6.5.2 IBM Network Station Boot Configuration

Before you boot an IBM Network Station, ensure that the BOOTPD or DHCPD, TFPTD and NSLD servers have been properly initialized and are operational. Specifically, the IBM Network Station requires that the TFTPD server run with the XFERMODE OCTET command option in effect, so that it will not attempt to translate files that are requested as NETASCII. It is also necessary to specify the CREATION /QIBM command option to allow the Network Station Manager to create or update configuration files using TFTP.

11.7 Java Programs on the IBM Network Station

The only application programming capability available on the IBM Network Station is provided through the Java language and the Java virtual machine. Two fundamentally different forms of Java programs may be written for use on a Network Station:

Applets are small, reusable components that run in conjunction with, and under the control of the IBM Network Station Web browser.

Applications are complete, stand-alone programs that are loaded either during Network Station startup or later by launching the application from the menu bar on the IBM Network Station.

In either case, the executable program, in the form of one or more Java class files, is downloaded from a server via either NFS or TFTP, depending on how your network is set up.

Note: Release 2.5+ of the Network Station client code permits only one Java program, either application or applet, to be executed at a time. This restriction may be lifted in a later release.

11.8 Setting up to Run Java and NetRexx Programs

When running a Java or NetRexx program on the IBM Network Station, the appropriate class libraries must reside on directories that are known and mounted on the local file system. This is not a problem for the Java classes because the Network Station client code distribution includes the Java class library in both zipped and unzipped format.

The NetRexx classes, however, are not shipped with the Network Station client code. To make it possible to run the **GuiMon** sample application or any other NetRexx program, the NetRexx class library must be visible in the local file system of the Network Station.

For our work during the residency, we chose to copy the NetRexx classes from the NetRexx file space structure to the Network Station client code file space structure /QIBM/ProdData/ that already contained the Java classes. The reason is that we did not want to create an additional mount point for the NetRexx standard classes, instead keeping all standard classes in one base file structure. Copying the NetRexx classes into the /QIBM/ProdData/ structure is also a good investment from the standpoint of performance; see 11.10.1, “Performance Considerations” on page 133 for further discussion of this subject.

Since the installation and service procedure for the Network Station client code completely replaces the /QIBM/ProdData/ structure, the copying of the NetRexx classes must be redone each time the client code is serviced or refreshed.

11.8.1 How to Copy the NetRexx Runtime Environment

You can copy the NetRexx classes to the Network Station client code BFS structure using either the shell or standard CMS facilities. In the following sections, we illustrate both methods. Of course, in either case your user ID must have write access to the QIBM file space.

11.8.1.1 Copying the NetRexx Classes with the Shell

After starting up the shell, use the **mkdir** command to create the netrexx/lang directory structure in the java/classes subdirectory, and then copy the NetRexx classes into it with the **cp** command. Figure 35 on page 129 illustrates how we did this on our test system. If you have changed the default installation of either NetRexx or the Network Station client code, then you will may have to alter the directory specifications accordingly.

```
Ready; T=0.01/0.01 15:13:51
openvm mount /../VMBFS:VMSYS:ROOT/ /
Ready; T=0.01/0.02 15:13:57
openvm shell

IBM
Licensed Material - Property of IBM
5654-030 (C) Copyright IBM Corp. 1995
(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.

maint:/: >
cd /QIBM/ProdData/NetworkStation/java/classes
maint:/QIBM/ProdData/NetworkStation/java/classes: >
mkdir -p netrexx/lang
maint:/QIBM/ProdData/NetworkStation/java/classes: >
cp /usr/NetRexx/netrexx/lang/*.* netrexx/lang
maint:/QIBM/ProdData/NetworkStation/java/classes: >
```

Figure 35. Copy the NetRexx Runtime Environment Using the Shell

11.8.1.2 Codepages - ASCII <> EBCDIC for Network Stations

The Java class library shipped with the Network Station client code does not include the two class files (ByteToCharCp1047.class and CharToByteCp1047.class) required to support the EBCDIC codepage 1047 used by OpenEdition. Thus, you will want to copy these two class files from the sun/io subdirectory of your VM Java class library to the corresponding part of your Network Station client code structure as well. Figure 36 shows how this can be done in the shell environment, continuing from where we left off at the end of Figure 35.

```
maint:/QIBM/ProdData/NetworkStation/java/classes: >
cp /usr/java/classes/sun/io/*Cp1047.class sun/io
maint:/QIBM/ProdData/NetworkStation/java/classes: >
```

Figure 36. Copy the Codepage 1047 Class Files Using the Shell

See Chapter 7, “Code Pages - ASCII <> EBCDIC Issues” on page 81 for further information on codepage issues.

11.8.1.3 Copying the NetRexx and Cp1047 Classes without the Shell

Figure 37 on page 130 displays the NETCPY EXEC, included with the sample program set for this redbook, which uses the CMS OPENVM and PIPE commands to accomplish the same operations shown in Figure 35 and Figure 36 without the shell.

```
/********************************************************************/
/* Use this exec to create directories java/classes/netrexx/lang and */
/* to copy the standard NetRexx classes and the Cp1047 code page      */
/* classes to the Network Station client code BFS file structure.     */
/********************************************************************/
address command
'EXEC OPENVM CREAT DIR /QIBM/ProdData/NetworkStation/java/classes/netrexx'
'EXEC OPENVM CREAT DIR /QIBM/ProdData/NetworkStation/java/classes/netrexx/lang'
'PIPE bfsdirectory /usr/NetRexx/netrexx/lang | spec w1 | stem file.'
do i=1 to file.0
  'PIPE bfs /usr/NetRexx/netrexx/lang/'file.i,
  '| bfs /QIBM/ProdData/NetworkStation/java/classes/netrexx/lang/'file.i
end
'PIPE bfs /usr/java/classes/sun/io/ByteToCharCp1047.class',
  '| bfs /QIBM/Prodata/NetworkStation/java/classes/sun/io/ByteToCharCp1047.class'
'PIPE bfs /usr/java/classes/sun/io/CharToByteCp1047.class',
  '| bfs /QIBM/Prodata/NetworkStation/java/classes/sun/io/CharToByteCp1047.class'
exit
```

Figure 37. NETCPY EXEC to Copy the NetRexx and Cp1047 Classes

11.9 Starting a Java or NetRexx Program on your IBM Network Station

In the following sections we explain how to run the demonstration application GuiMon (described in detail in Chapter 10, "The GUIMON Sample Program" on page 101 and included in the sample program set for this redbook) on an IBM Network Station.

The client side of GuiMon is a stand-alone Java application which may be executed by the IBM Network Station Java Virtual Machine (JVM). The JVM is in charge of controlling the Java execution environment and obtaining resources from the Network Station kernel. To start a Java application you must call the JVM and pass to it a series of parameters such as the name of the initial class of the application. Since this might be a rather long and complicated command string, you will find it more convenient to define a Menu Bar item to allow the application to be started by simply clicking the appropriate button on the Menu Bar. The procedure for doing this is described in 11.11, "Using NSM to Add a Java Application to the Menu Bar" on page 133.

The class files comprising the application must be installed in a server file system that is accessible to the Network Station kernel. Since you will want to keep application class files separate from the standard Java and NetRexx class libraries, you will in general have to tailor the local file system of the IBM Network Station to define the additional server file structures that are to be mounted by the kernel. We explain how to do this file system tailoring in 11.10, "How to Tailor the Local File System" on page 131.

In a Network Computing world, applications may be stored on many different hosts, not just the one used as a boot server. In fact, the class files used by a single Java or NetRexx application could reside on multiple VM/ESA servers. To show you how this can be done, we will use GuiMon as an example.

The client side of the GuiMon application consists of eleven class files. To show how resources can be shared among different systems we installed:

- The basic GuiMon class files in directory /home/demo on VM1

```
GuiMon.class                      GuiMonMenuAction.class
GuiMonActions.class               GuiMonTimeScroll.class
GuiMonFrame.class                 GetPerf.class
GuiMonFrameController.class       TimeGraph.class
```

- The AboutFrame class files in directory /home/demo on VM2

```
AboutActionClass.class
AboutFrame.class
AboutFrameController.class
```

where VM1 is a VM system in Germany and VM2 is a VM system in Sweden. Our network configuration is illustrated in Figure 38.

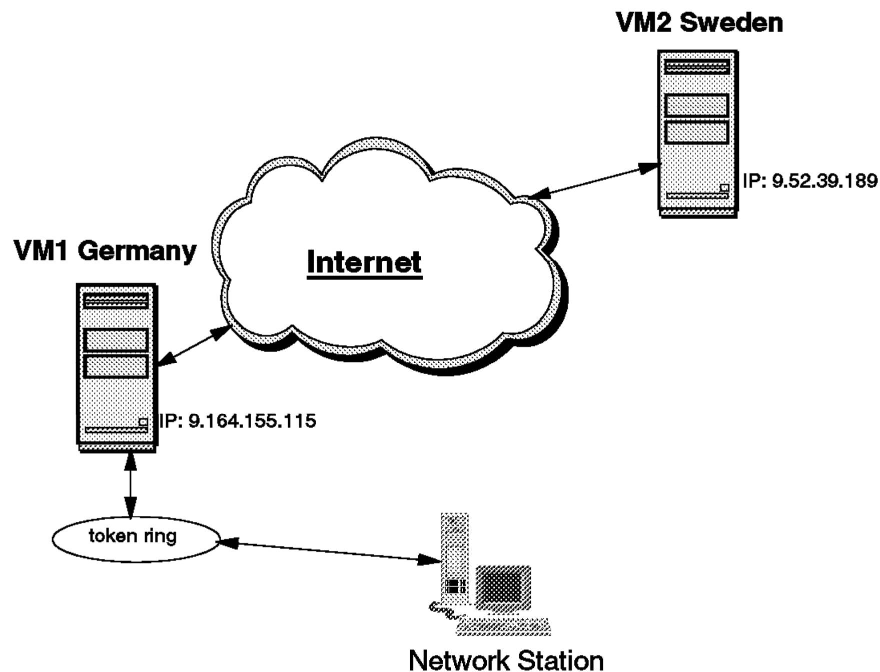


Figure 38. Network Configuration

11.10 How to Tailor the Local File System

The kernel running in the Network Station needs a file system structure from which to serve applications′ file requests. This structure is known as the local file system but effectively consists of mounts of remote directories since the IBM Network Station has no local file storage of any kind. Many directories are automatically mounted during the initialization process including:

- All of the font directories
- The keyboard definitions directory
- The loadable modules directory

The file-service-table parameter allows you to specify any number of additional directories on any hosts that are to be mounted on the local file structure. Each entry in the list is composed of the following positional parameters:

local-unix-mount-point The UNIX style local name for this file service access point.

local-vms-mount-point This value should always be **nil**.

server The symbolic name or the IP address of the host.

protocol For S/390 hosts, either **tftp** or **nfs**.

server-mount-point The name of the service access point on the file server.

file-name-type We recommend you always speciy **unix**.

retransmission-timeout The amount of time in seconds between successive transmissions of a file service request.

transaction-timeout The amount of time in seconds to attempt a file service request before a failure situation is declared.

read-size The amount of data in bytes to be requested in a single read request. If you experience problems reading files across gateways, try decreasing this parameter to 1024 for nfs or 512 for tftp.

write-size The amount of data in bytes to be requested in a single write request. If you experience problems writing files across gateways, try decreasing this parameter to 1024 for nfs or 512 for tftp.

In order to make our Network Station be able to run the GuiMon application we need to create a file service table. This operation is not supported by the current release of the NSM, and thus the configuration file in which the file service table is defined must be edited manually. The file that has to be altered is one of the special files called defaults.dft that resides in the file structure in the /QIBM/UserData/NetworkStation/StationConfig subdirectory.

IBM Network Station configuration files are stored in ASCII and thus cannot be directly edited; an exec called ASCXED is provided to temporarily translate an ASCII file into EBCDIC for editing, then translate it back to ASCII after it is saved.

Use ASCXED to tailor the file by entering:

ASCXED /QIBM/UserData/NetworkStation/StationConfig/defaults.dft

Our file service table to support distributed GuiMon looked like this:

```
This file service table is for the host in Germany
set file-service-table[-1] = {
  "/VM1/" nil 9.164.155.115 tftp
    "/home/demo/" unix 6 30 512 512 }

This file-service-table is for the host in Sweden
set file-service-table[-1] = {
  "/VM2/" nil 9.52.39.189 tftp
    "/home/demo/" unix 6 30 512 512 }
```

Notes:

1. Each mount point needs its own file-service-table[-1] entry because the environment variable file-service-table can support only one mount point. This might be changed in a later release.
2. The subscript [-1] is very important; it informs actlogin that the file-service-table entries are to be appended to its list.

11.10.1 Performance Considerations

When planning to share an application class library between two systems, or simply to have (as we strongly recommend) the standard class library and the application's classes on different directories in one system, it is important to keep in mind how the Java Virtual Machine in the Network Station will request resources from the Network Station kernel. When you issue the Java command, JVM initialization involves loading a large number of class files that are part of the standard Java class library. Subsequently, the first application class file is loaded, and during its execution additional class files will be loaded as required. In all cases, the JVM searches each directory path in the current ***classpath***, in the order in which they are specified, until the requested class file is found.

For example, if the JVM is invoked as follows:

```
java -classpath /VM2:/VM1:/QIBM/ProdData/NetworkStation/java/classes ...
```

then each time a class file must be read, whether it is a standard Java or NetRexx class or an application class, the JVM will look for it first in directory path /VM2, then in /VM1, and finally in the path containing the standard Java library. In our case, since /VM2 is a mount point for a file system on a remote host, this means that every request to load a class file is going to require a network interaction to interrogate the remote file system.

From the network point of view, the search order is very important. Since the vast majority of the class files that will be used during the execution of most Java or NetRexx applications will come from the Java class library, not from the application class libraries, the classpath in the example should be specified like this to improve performance:

```
Java -classpath /QIBM/ProdData/NetworkStation/java/classes:/VM1:/VM2 ...
```

11.11 Using NSM to Add a Java Application to the Menu Bar

The IBM Network Station Manager program is a browser based application program with which you may perform many of the setup and management tasks that are associated with administering IBM Network Stations and supporting Network Station users. If you are unfamiliar with the basic function and operation of the NSM, we refer you to the description in the redbook *S/390 - IBM Network Station - Getting Started*, SG24-4954, for a thorough introduction. In the remainder of this section, we describe how to use the NSM to add a Java Application Menu Item.

To add the Java application GuiMon to your Network Station Menu Bar follow these steps:

- Type the NSM administrator user name and password on the login screen shown in Figure 39 on page 134.

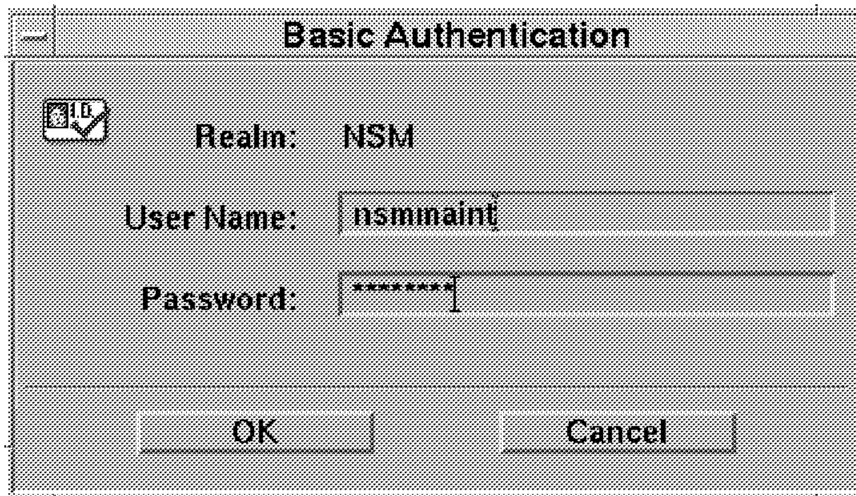


Figure 39. IBM Network Station Manager Login Screen

- Click on **OK** to reach the administrator main menu shown in Figure 40.

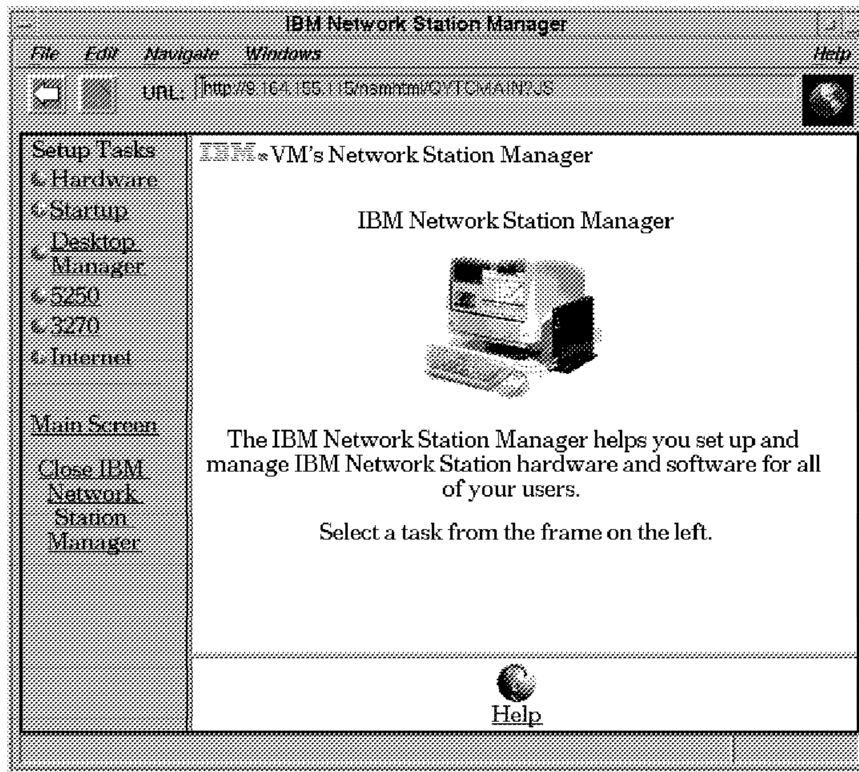


Figure 40. IBM Network Station Manager Administrator Main Menu

- Click on **Startup** in the Setup Tasks list.
- Click on **Menus** in the expanded Startup sublist.
- Select User Defaults and type in a user name. We used **perfmon**, as can be seen in Figure 41 on page 135.

Notes:

1. The name is automatically folded to upper case by the NSM.
2. A corresponding user ID must exist in the CP user directory of the VM/ESA boot server in order for the NSLD to perform authentication at Network Station login time.

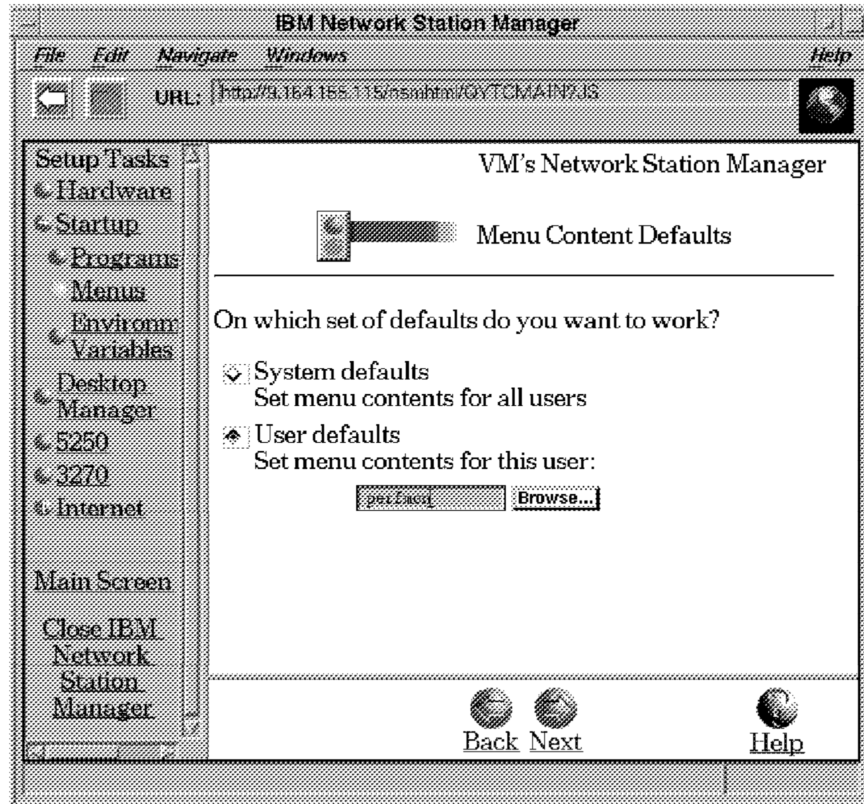


Figure 41. IBM Network Station Manager Startup Menus

- Click on **Next**.
- Scroll down to Java Application Menu Items.
- In the “Menu item label” field, type **GuiMon**.
- In the “Application (class) name” field, also type **GuiMon**.
- In the “Class path” field, type the following:

  /QIBM/ProdData/NetworkStation/java/classes:/VM1:/VM2

  as illustrated in Figure 42 on page 136.

  Points to Remember:

  1. The class path is a list of all the directory paths to the classes needed by the application, including the standard Java class library. Each time the JVM must load a class file, these directory paths are searched in the order specified.
  2. In our example, we have defined VM1 and VM2 as mount points in the file service table (see 11.10, “How to Tailor the Local File System” on page 131 for details). VM1 defines the location of the GuiMon classes on

the local VM server, and VM2 identifies the directory path to the About Frame classes on a remote VM server.

3. To reduce network traffic and improve responsiveness, it is very important to order the directory path specifications in the class path list so as to minimize the likely search time for a class file. In general this means that the standard Java class library should be listed ahead of the application class libraries, and that directories on nearby servers should be listed before those on more remote servers. See 11.10.1, "Performance Considerations" on page 133 for more information..

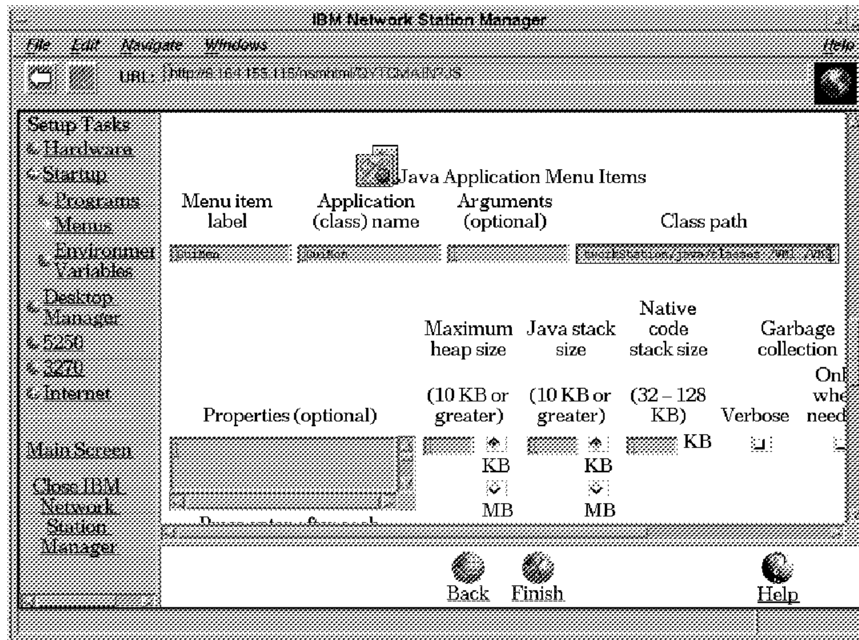


Figure 42. IBM Network Station Manager Java Application Menu Items

- Click on the **Finish** icon.
- Click on the **Back** icon.
- Click on **Close IBM Network Station Manager**.

11.12 Starting the GuiMon Application on the Network Station

GuiMon is a client/server application that exploits the Java and NetRexx execution capabilities of both the IBM Network Station and VM/ESA V2R3M0 to illustrate the potential utility of this powerful combination of platforms. By installing the GuiMon client code in two packages, the GuiMon classes on the VM/ESA boot server in Germany and the AboutFrame classes on a separate VM/ESA server in Sweden, we further demonstrate how flexible network computing can be and how the Network Station is a major player in this environment. The configuration work described above in 11.10, "How to Tailor the Local File System" on page 131 and 11.11, "Using NSM to Add a Java Application to the Menu Bar" on page 133 means that the details of the network configuration are transparent to the end user.

11.12.1 Login to the Network Station

When the IBM Network Station has completed downloading and booting the kernel, or when the previous user of a running Network Station has logged out, the Network Station Login Screen is displayed. Type the user ID (**perfmon** in our example, as shown in Figure 43) and press Enter or click on **OK**.

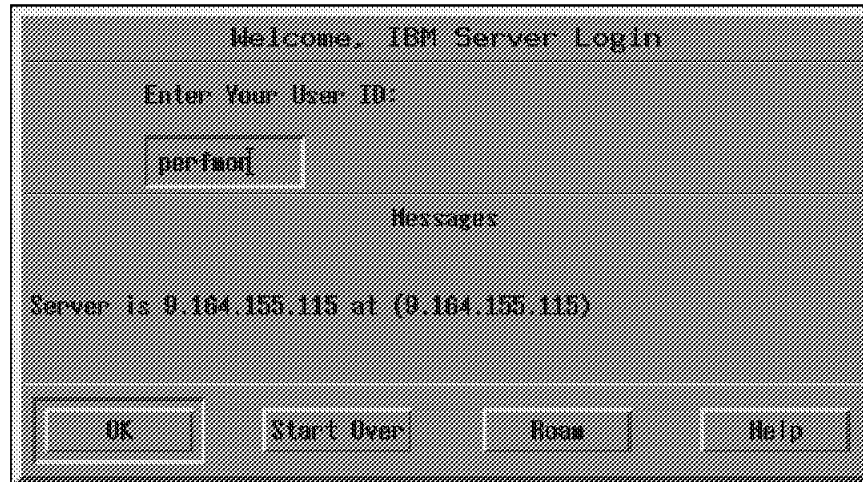


Figure 43. IBM Network Station Login Screen

On the next screen type the password and again press Enter or click on **OK**. As password validation proceeds, the NSLD displays progress messages; when validation is complete, and the message "Authentication completed successfully" appears, click on **Done** to allow login to continue.

11.12.2 Start GuiMon from the Menu Bar

When the login process has completed the initialization of the user's Network Station environment, based on the configuration information previously saved by the NSM, the Menu Bar will appear at the bottom of the screen. Click on the GuiMon button to start the GuiMon application.

First the JVM initializes, requiring the loading of a large number of class files from the standard Java class library. When JVM initialization is complete, the main application class file, GuiMon.class, is requested and found in directory path /VM1. During its initialization, GuiMon requires additional class files which are also located in and loaded from /VM1. When GuiMon is ready to accept requests, it displays the window shown in Figure 44 on page 138.

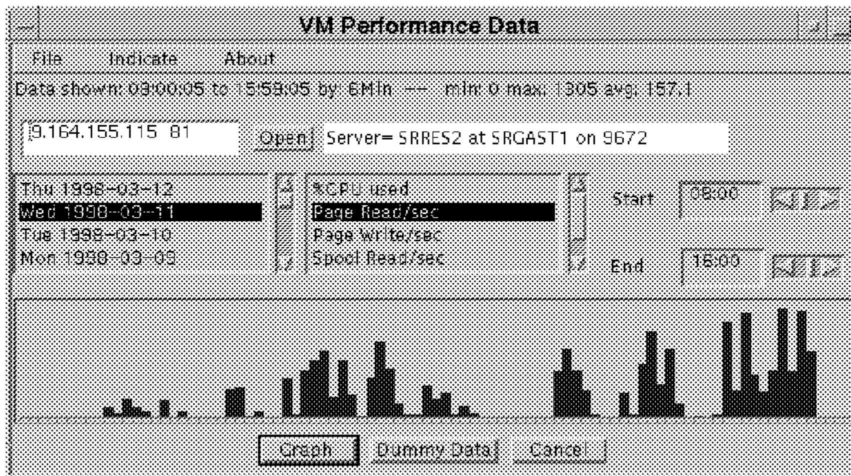


Figure 44. Built by GuiMon Classes Loaded from Germany

Up to this point the AboutFrame class files located in directory path /VM2 have not been required. If we now click on About in the action bar, and then select AboutFrame from the resulting pulldown menu, the necessary AboutFrame classes are downloaded from the system in Sweden, producing the display illustrated in Figure 45.

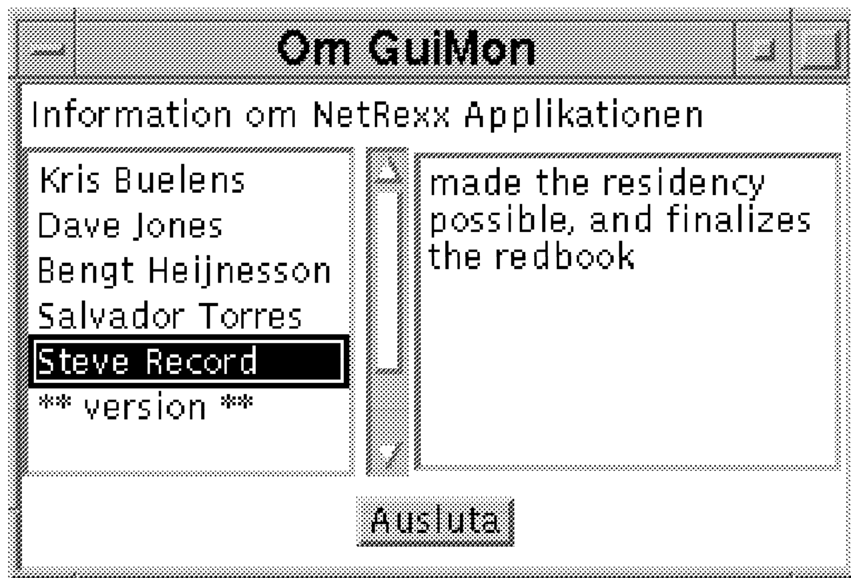


Figure 45. Built by the AboutFrame classes Loaded from Sweden

To make the international distinction more apparent, we have translated the AboutFrame title, action bar text, and push button label, all of which are coming from the VM2 system, into Swedish (compare to the original version shown in Figure 9 on page 55); but the data in the list boxes, derived from VM1, remains English.

11.12.3 Summary

This very simple example of distributing the parts of an application across multiple network-connected systems demonstrates, in part by its very simplicity, how flexible network computing can be, how powerful the Network Station can be, and above all how readily VM/ESA can be a major player in this style of network computing, acting as both a boot and an application server to Network Stations.

Appendix A. Frequently Asked Questions

Here is a list of questions you might have, or problems you might encounter.

A.1 NullPointerException - General Problem

A NullPointerException is most of the time caused by a program trying to use a not initialized variable. An example, reading a file:

```
   loop until line = null
1       line = String(SomeFile.readLine())
2       Say 'read=' line
   end
```

When at **1** no more records exist in the file, variable `line` becomes undefined, or null. Then the execution of **2** results in a NullPointerException. The next alternatives avoid the problem:

```
loop forever
     line = String(SomeFile.readLine())
     if line = null then leave
     Say 'read=' line
end
⋮
line = String(SomeFile.readLine())
loop while line<>null
     Say 'read=' line
     line = String(SomeFile.readLine())
end
```

A.2 NullPointerException - With Compound Variables

In 4.3.8, "Stems - Array Variables - Indexed Strings" on page 50 it has been mentioned that before using a NetRexx array, the stem has to get a value assigned to it. In OO terms: the stem has to be an object already. If this is forgotten, the result can be a NullPointerException.

For example, suppose that both `place` and `loc` are variables that have been assigned a value, but that `description` is a variable that has only been declared. At execution time the following error is seen:

```
  8 *=*  description = Rexx              -- Declare it as a property
  ⋮
 19 *=*  place='Boeblinger MineralTerme'
 20 *=*  loc='nice'
  ⋮
 31 *=*  description[place]=loc
java.lang.NullPointerException
        at testsubf.main(testsubf.nrx:31)
```

Initially one is very confused and thinks that either the variable `loc` or the variable `place` is undefined. One just tends to forget that the cause of the error can be that the variable `description` is not initialized.

The solution is to create the stem variable explicitly by inserting `description=''` before line 31.

© Copyright IBM Corp. 1998

A.3 NetRexx: No Data Type Problems Anymore?

Even though NetRexx does indeed hide many data type problems, sometimes it still fails, as shown below.

A program, GuiMon, contained this statement to obtain the string selected in a list box.

```
Afile = theFrame.lstFile.GetSelectedItem()  -- Selected file
```

At compile time an error is seen:

```
308 +++  Afile = Afile.word(2).Translate(' ','-').space(0) -- Keep only
    +++               ┬┬┬┬
    +++ Error: The method 'word(byte)' cannot be found in class
               'java.lang.String' or a superclass
```

This means that NetRexx assigned data type "String" to variable "Afile." The solution is simple: explicitly set a data type when "Afile" is first used:

```
Afile = Rexx theFrame.lstFile.GetSelectedItem()  -- Selected file
```

A.4 Error Messages Not Always Very Accurate

One of the things you have to keep in mind is that error messages seldom give a detailed reason. This may set you on the wrong track when searching for a solution.

For example: "Can't find class file." This error message is issued both when

- the file cannot be found in the classpath, and
- the class file is unusable.

The latter case can be caused by the class file having been accidentally translated to EBCDIC. Class files must be ASCII. When transporting them between platforms, be sure to transport them as BINARY files.

A.5 File Not Found

This a variation of A.4, "Error Messages Not Always Very Accurate."

A.6 Threads Class Not Found

When Java refuses to start with an error message as shown in the example below, verify the CLASSPATH definition

```
EXEC JAVA javatest
Unable to initialize threads: cannot find class java/lang/Thread
```

The CLASSPATH must reflect the directories in which Java and NetRexx are installed, refer to 3.8.1, "Important Environment Variables" on page 33 for more information about CLASSPATH.

A.7 External Link Files Not Found

When an external link is created to a CMS file, “file not found” conditions may be encountered if the &&& option is not specified on the OPENVM CREATE EXTLINK command.

A.8 Reading Java Abend Messages

Here is a Java abend message caused by a misbehaving NetRexx program:

```
java.lang.NullPointerException
        at ServerHandler.ListFile(ServerHandler.nrx:101)
        at ServerHandler.runUser(ServerHandler.nrx:24)
        at TCPServer.run(TCPServer.nrx:56)
        at TCPServer.run(TCPServer.nrx:43)
        at java.lang.Thread.run(Thread.java)
```

The error here is a `NullPointerException`. Java prints a “traceback” of the methods active at the time. Most of the time the first line gives enough information to correct the problem.

In our case, `ServerHandler.ListFile(ServerHandler.nrx:101)` means that the problem occurred in source file ServerHandler.nrx at line 101.

We found that sometimes the problem is caused by what is mentioned on the last line of the printout. As has been said, NetRexx can convert data types, and when this fails the first line tells what NetRexx was trying to do.

Here is an example; a NetRexx trace shows what tried to execute.

```
 25 *=*  z='sss'
 26 *=*  Say 'Mineraltherme in Boeblingen are cool'.word(z)
java.lang.NumberFormatException: Not a number
        at netrexx.lang.Rexx.intcheck(Rexx.nrx:832)
        at netrexx.lang.Rexx.subword(Rexx.nrx:1255)
        at netrexx.lang.Rexx.word(Rexx.nrx:1303)
        at testsubf.main(testsubf.nrx:26)
```

In the above case, it is clear that “z” has to be converted to an integer. Sometimes though, it is not obvious that a data type conversion is required (as REXX programmers we are no longer used to data types). Reading the whole traceback printout, bottom up, reveals what NetRexx is trying to do.

A.9 Virtual Storage Requirements

A very important fact to remember is that Java cannot automatically use all the DEFINEd STORAGE of a virtual machine.

Java provides two options to control virtual storage use.

```
  -ms<number>       set the initial Java heap size
  -mx<number>       set the maximum Java heap size
```

With the current VM/ESA (and OS/390) Java implementation, “ms” and “mx” are treated equally, and all the memory is taken at once, in one big piece.

When compiling the GuiMon NetRexx program, we soon had out-of-memory errors.

It appears that the version of Java used during this residency had a default of mx value that is too small to compile some NetRexx programs. Therefore, we updated our NRC EXEC to specify a 4MB mx value:

```
'EXEC OPENVM RUN /usr/java/openvm/java -mx4m ...
```

By the time that Java and NetRexx become generally available on VM, the default mx value for the NetRexx compiler may have been changed to 8MB.

Beware: As on VM Java allocates the whole mx value at once, in one big piece, specifying an mx value that is too high is not always good either. Remember that when a CMS user has a DEFINEd STORAGE of 32M, not all 32 megabytes are available. Usually quite a few saved segments are located between 16M and 32M.

On our test system the default storage allocation looks like:

```
 q nss name cms map
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMG
0122 CMS      NSS      0000256K 00000  0000D   EW  A  00033   00-15     N
                                00020  00023   EW
                                00F00  013FF   SR
Ready;
 q segment
Space     Name      Location  Length    Loaded   Attribute
CMSVMLIB  DMSRTSEG  01796B00  0003FBE0  YES      SYSTEM
CMSVMLIB  VMLIB     01700000  00096AC0  YES      SYSTEM
CMSPIPES  RXSOCKET  0188F498  0000FC88  YES      SYSTEM
CMSPIPES  PIPES     01800000  0008F458  YES      SYSTEM
INSTSEG   CMSINST   01400000  001C57E0  YES      SYSTEM
Ready;
 q v stor
STORAGE = 0032M
Ready;
```

You can see above that CMS itself is located from 15M through 19M, other saved segments are located at 20M, 23M, and 24M. When taking into account that CMS very soon uses dome storage in the upper part of virtual storage, it can be clear the biggest piece we can ever get is 7MB.

Therefore, we recommend a DEFINEd STORAGE that is bigger than 32M for NetRexx program development, for example 64M.

A.10 Killing the Java Virtual Machine in VM

Sometimes it can be required to stop the Java virtual machine on VM, some servers cannot be stopped otherwise. An easy choice that always works obviously is entering:

```
#CP IPL
```

This has as side effect that you lose all ACCESSed disks and so on. Therefore, it is useful to know two other possibilities. Enter:

```
¢c         cancel
¢v         cancel and produce a minidump
```

Note, the ¢ character stands for X′4A′. If your keyboard has no key for it, update your PROFILE EXEC to include, for example:

```
′CP SET PF02′ ′4A′x
```

For more information on OpenEdition terminal control sequences, see 6.2.1, “Useful Control Sequences” on page 77.

A.11 Runtime Problems

A couple of remarks may help you to avoid some problems we encountered.

- Running Java programs under CMS Rel 13 does not work. You need CMS Rel 14 (the one coming with VM/ESA 2.3.0) and a recent maintenance level. Similar remarks apply to the LE/370 library: use the one delivered with VM/ES 2.3.0, and apply the latest fixes.

  When using backlevel versions, one of the observed failures is a CPU loop.

- The SCEERUN LOADLIB must be included in the GLOBAL LOADLIB. Failure to do so may simply result in a non-zero return code being displayed, as in the example below:

  ```
  openvm run /usr/java/openvm/java -version
  Ready KRIS at SRGAST1 (01255); T=0.17/0.28 18:24:35
  ```

A.12 Installation Problems

A couple of times the installation of Java failed. When having problems please verify the following:

- The installer virtual machine needs a large amount of virtual storage (for example, CP DEFINE STORAGE 200M).
- The installer virtual machine needs to be an SFS administrator: the installation of Java and NetRexx requires the enrollment of new SFS users _JAVA and _NETREXX.
- The BFS must be initialized with some basic users and directories (“root” for example). This also means that “ROOT” must be present in the CP directory (see 2.7, “Installing Java and NetRexx without the Shell and Utilities” on page 16).
- Using the LE/370 library delivered with previous versions of VM/ESA results in CMS abends during installation (protection or addressing exceptions).
  - Ensure that the Y-disk is the one coming with VM/ESA 2.3.0.
  - Also verify that the saved segments SCEE and SCEEX correspond to the level of VM/ESA 2.3.0. This remark is very important to installations that for example run CMS Level 13 alongside the new CMS Level 14. In that case it is very likely that the SCEE and SCEEX saved segments are those of VM/ESA 2.2.0 instead of those of VM/ESA 2.3.0. A bypass is simple: remove the entries for SCEE and SCEEX from the SYSTEM SEGID on CMS 14′s S-disk (remove the PSEG and dependent LSEG cards).
- We believe that the installation of Java and NetRexx in the BFS may fail when the SFS server has to perform a control data backup during the installation. Therefore it may be wise to issue:

  ```
  CP SET SECUSER VMSERVx *
  CP SEND VMSERVx BACKUP
  ```

before starting the installation. Keeping the secondary user ID set to the installer user ID during the installation is good as well. It will allow you to easily spot problems in the server, such as disk full or control data backups.

Appendix B. NetRexx Language Quick Start

For your convenience we reproduce the NetRexx introduction text that can be found as "NROVER.DOC" amongst the files delivered with the NetRexx software. It has been written for NetRexx level 1.125.

Remember that Redbook *Creating Java Applications Using NetRexx*, SG24-2216 is a good starting point to write NetRexx applications. It has been written for OS/2 but applies to nearly all platforms where NetRexx can be used.

B.1 Introduction

NetRexx is a new programming language derived from both REXX and Java; it is a dialect of REXX that retains the portability and efficiency of Java, while being as easy to learn and to use as REXX.

NetRexx is an effective alternative to the Java language. With NetRexx, you can create programs and applets for the Java environment more easily than by programming in Java. Using Java classes is especially easy in NetRexx as you rarely have to worry about all the different types of numbers and strings that Java requires.

This document summarizes the main features of NetRexx, and is intended to help you start using it quickly. It's assumed that you have some knowledge of programming in a language such as REXX, C, BASIC, or Java, but a knowledge of "object-oriented" programming isn't needed.

This is not a complete tutorial, though -- think of it more as a "taster"; it covers the main points of the language and shows some examples you can try or modify.

For more samples (and examples of using Java classes from NetRexx), and for more formal details of the language, please see the other NetRexx documents at URL

```
http://www2.hursley.ibm.com/netrexx/
```

You'll find the NetRexx software to download there, too.

B.2 NetRexx Programs

The structure of a NetRexx program is extremely simple. This sample program, "toast," is complete, documented, and executable as it stands:

```
/* This wishes you the best of health. */
say 'Cheers!'
```

This program consists of two lines: the first is an optional comment that describes the purpose of the program, and the second is a SAY statement. SAY simply displays the result of the expression following it -- in this case just a literal string (you can use either single or double quotes around strings, as you prefer).

© Copyright IBM Corp. 1998

To run this program, edit a file called toast.nrx and copy or paste the two lines above into it. You can then use the NetRexxC Java program to compile it, and the java command to run it:

```
java COM.ibm.netrexx.process.NetRexxC toast
java toast
```

You may also be able to use the NETREXXC command to compile and run the program with a single command (details may vary -- see the installation and user's guide document):

```
netrexxc toast -run
```

Of course, NetRexx can do more than just display a character string. Although the language has a simple syntax, and has a small number of statement types, it is powerful; the language allows full access to the rapidly growing collection of Java programs known as "class libraries," and allows new class libraries to be written in NetRexx.

The rest of this document introduces most of the features of NetRexx. Since the economy, power, and clarity of expression in NetRexx is best appreciated with use, you are urged to try using the language yourself.

B.3 Expressions and Variables

As with SAY in the "toast" example, many statements in NetRexx include *expressions* that will be evaluated. NetRexx provides arithmetic operators (including integer division, remainder, and power operators), several concatenation operators, comparison operators, and logical operators. These can be used in any combination within a NetRexx expression (provided, of course, that the data values are valid for those operations).

All the operators act upon strings of characters (known as *REXX strings*), which may be of any length (typically limited only by the amount of storage available). Quotes (either single or double) are used to indicate literal strings, and are optional if the literal string is just a number. For example, the expressions:

```
'2' + '3'
'2' + 3
  2 + 3
```

would all result in "5."

The results of expressions are often assigned to *variables*, using a conventional assignment syntax:

```
var1=5           /* sets var1 to '5'    */
var2=(var1+2)*10 /* sets var2 to '70' */
```

You can write the names of variables (and keywords) in whatever mixture of uppercase and lowercase that you prefer; the language is not case-sensitive.

This next sample program, "greet," shows expressions used in various ways:

```
/* A short program to greet you.                   */
/* First display a prompt:                         */
say 'Please type your name and then press ENTER:'
answer=ask             /* Get the reply into ANSWER */

/* If no name was entered, then use a fixed greeting, */
/* otherwise echo the name politely.               */
```

```
if answer='' then say 'Hello Stranger!'
             else say 'Hello' answer'!'
```

After displaying a prompt, the program reads a line of text from the user ("ask" is a keyword provided by NetRexx) and assigns it to the variable ANSWER. This is then tested to see if any characters were entered, and different actions are taken accordingly; if the user typed "Fred" in response to the prompt, then the program would display:

```
Hello Fred!
```

As you see, the expression on the last SAY (display) statement concatenated the string "Hello" to the value of variable ANSWER with a blank in between them (the blank is here a valid operator, meaning "concatenate with blank"). The string "!" is then directly concatenated to the result built up so far. These unobtrusive operators (the *blank operator* and abuttal) for concatenation are very natural and easy to use, and make building text strings simple and clear.

The layout of statements is very flexible. In the "greet" example, for instance, the IF statement could be laid out in a number of ways, according to personal preference. Line breaks can be added at either side of the THEN (or following the ELSE).

In general, statements are ended by the end of a line. To continue a statement to a following line, you can use a hyphen (minus sign) just as in English:

```
say 'Here we have an expression that is quite long, so' -
    'it is split over two lines'
```

This acts as though the two lines were all on one line, with the hyphen and any blanks around it being replaced by a single blank. The net result is two strings concatenated together (with a blank in between) and then displayed.

When desired, multiple statements can be placed on one line with the aid of the semicolon separator:

```
if answer='Yes' then do; say 'OK!'; exit; end
```

(Many people find multiple statements on one line hard to read, but sometimes it is convenient.)

B.4 Control Statements

NetRexx provides a selection of *control* statements, whose form was chosen for readability and similarity to natural languages. The control statements include IF... THEN... ELSE (as in the "greet" example) for simple conditional processing:

```
if ask='Yes' then say "You answered Yes"
             else say "You didn't answer Yes"
```

SELECT... WHEN... OTHERWISE... END for selecting from a number of alternatives:

```
select
  when a>0 then say 'greater than zero'
  when a<0 then say 'less than zero'
  otherwise say 'zero'
  end
```

DO... END for grouping:

```
if a>3 then do
  say 'A is greater than 3; it will be set to zero'
  a=0
  end
```

and LOOP... END for repetition:

```
loop i=1 to 10    /* repeat 10 times; I changes from 1 to 10 */
  say i
  end
```

The LOOP statement can be used to step a variable TO some limit, BY some increment, FOR a specified number of iterations, and WHILE or UNTIL some condition is satisfied. LOOP FOREVER is also provided. Loop execution may be modified by LEAVE and ITERATE statements that significantly reduce the complexity of many programs.

B.5 NetRexx Arithmetic

Character strings in NetRexx are commonly used for arithmetic (assuming, of course, that they represent numbers). The string representation of numbers can include integers, decimal notation, and exponential notation; they are all treated the same way. Here are a few:

```
'1234'
'12.03'
'-12'
'120e+7'
```

The arithmetic operations in NetRexx are designed for people rather than machines, so are decimal rather than binary, do not overflow at certain values, and follow the rules that people use for arithmetic. The operations are completely defined by the ANSI standard for REXX, so correct implementations will always give the same results.

An unusual feature of NetRexx arithmetic is the NUMERIC statement: this may be used to select the *arbitrary precision* of calculations. You may calculate to whatever precision that you wish, for financial calculations, perhaps, limited only by available memory. For example:

```
numeric digits 50
say 1/7
```

which would display

```
0.14285714285714285714285714285714285714285714285714
```

The numeric precision can be set for an entire program, or be adjusted at will within the program. The NUMERIC statement can also be used to select the notation (*scientific* or *engineering*) used for numbers in exponential format.

NetRexx also provides simple access to the native binary arithmetic of computers. Using binary arithmetic offers many opportunities for errors, but is useful when performance is paramount. You select binary arithmetic by adding the statement:

```
options binary
```

at the top of a NetRexx program. The language processor will then use binary arithmetic instead of REXX decimal arithmetic for calculations, throughout the program.

B.6 Doing Things with Strings

Another thing REXX is good for is manipulating strings in various ways. NetRexx provides the same facilities as REXX, but with a syntax that is more like Java or other similar languages:

```
phrase='Now is the time for a party'
say phrase.word(7).pos('r')
```

The second line here can be read from left to right as ″take the variable “phrase,” find the seventh word, and then find the position of the first “r” in that word″. This would display “3” in this case, because “r” is the third character in “party.”

In REXX, the second line above would have been written using nested function calls:

```
say pos('r', word(phrase, 7))
```

which is not as easy to read; you have to follow the nesting and then backtrack from right to left to work out exactly what′s going on.

In the NetRexx syntax, at each point in the sequence of operations some routine is acting on the result of what has gone before. These routines are called *methods*, to make the distinction from functions (which act in isolation). NetRexx provides (as methods) most of the functions that were evolved for REXX, for example:

- changestr (change all occurrences of a substring to another)
- copies (make multiple copies of a string)
- lastpos (find rightmost occurrence)
- left and right (return leftmost/rightmost character(s))
- reverse (swap end-to-end)
- space (pad between words with fixed spacing)
- strip (remove leading and/or trailing white space)
- pos and wordpos (find the position of string or a word in a string)
- verify (check the contents of a string for selected characters)
- word, wordindex, wordlength, and words (work with words)

These and the others like them, and the parsing described in the next section, make it especially easy to process text with NetRexx.

B.7 Parsing Strings

The previous section described some of the string-handling facilities available; NetRexx also provides REXX string parsing, which is a fast and simple way of breaking up strings of characters using simple pattern matching.

A PARSE statement first specifies the string to be parsed, often taken simply from a variable. This is followed by a *template* which describes how the string is to be split up, and where the pieces are to be put.

B.7.1 Parsing into Words

The simplest form of parsing template consists of a list of variable names. The string being parsed is split up into words (sequences of characters separated by blanks), and each word from the string is assigned (copied) to the next variable in turn, from left to right. The final variable is treated specially in that it will be assigned a copy of whatever is left of the original string and may therefore contain several words. For example, in:

```
parse 'This is a sentence.'   v1 v2 v3
```

the variable v1 would be assigned the value "This," v2 would be assigned the value "is," and v3 would be assigned the value "a sentence.."

B.7.2 Literal Patterns

A literal string may be used in a template as a pattern to split up the string. For example

```
parse 'To be, or not to be?'   w1 ','  w2 w3 w4
```

would cause the string to be scanned for the comma, and then split at that point; each section is then treated in just the same way as the whole string was in the previous example.

Thus, w1 would be set to "To be," w2 and w3 would be assigned the values "or" and "not," and w4 would be assigned the remainder: "to be?." Note that the pattern itself is not assigned to any variable.

The pattern may be specified as a variable, by putting the variable name in parentheses. The following statements:

```
comma=','
parse 'To be, or not to be?'   w1 (comma) w2 w3 w4
```

therefore have the same effect as the previous example.

B.7.3 Positional Patterns

The third kind of parsing mechanism is the numeric positional pattern. This works just like the string pattern except in syntax; it specifies a column number (which may be absolute or relative, and derived from a variable if necessary).

String patterns and positional patterns can be mixed.

B.8 Indexed Variables

NetRexx provides an indexed variable mechanism, adapted from the compound variables of REXX.

NetRexx string variables can be referred to simply by name, or also by their name qualified by another string (the *index*). When an index is used, a value associated with that index is either set or extracted; in the latter case, the initial value of the variable is returned if the index has not been used to set a value. For example, the program:

```
dognoise='bark'
dognoise['pup']='yap'
dognoise['bulldog']='grrrrr'
say dognoise['pup'] dognoise['terrier'] dognoise['bulldog']
```

would display

```
yap bark grrrrr
```

Any expression may be used inside the brackets; the resulting string is used as the index. Multiple dimensions may be used, if required:

```
dognoise='bark'
dognoise['spaniel', 'brown']='ruff'
say dognoise['spaniel', 'brown'] dognoise['terrier']
```

which would display

```
ruff bark
```

Here's a more complex example, a test program with a function (called a *constant method* in NetRexx) that removes all duplicate words from a string of words:

```
/* justonetest.nrx -- test the justone function.       */
say justone('to be or not to be')    /* simple testcase */
exit

/* This removes duplicate words from a string, and      */
/* shows the use of a variable (HADWORD) which is       */
/* indexed by arbitrary data (words).                   */
method justone(wordlist) constant
  hadword=0           /* show all possible words as new */
  outlist=''              /* initialize the output list */
  loop while wordlist\=''     /* loop while we have data */
    /* next, split WORDLIST into first word and residue */
    parse wordlist word wordlist
    if hadword[word] then iterate   /* loop if had word */
    hadword[word]=1   /* remember we have had this word */
    outlist=outlist word     /* add word to output list */
    end
  return outlist           /* finally return the result */
```

Running this program would display just the four words "to," "be," "or," and "not."

This example also uses the built-in *string parsing* provided by the PARSE statement. In this instance, the value of WORDLIST is parsed, with the first word of the value being assigned to the variable WORD and the remainder being assigned back to WORDLIST (replacing the original value).

[Author's note: since the notation for indexed variables looks just like arrays (see the next section), but does not suffer the restrictions of arrays, I like to call them *disarrays*.]

B.9 Arrays

NetRexx also supports Java's fixed-size *arrays*. These are an ordered set of items, indexed by integers. To use an array, you first have to construct it; an individual item may then be selected by an index whose value must be in the range 0 through N-1, where N is the number of items in the array:

```
array=String[3]           -- make an array of three Java Strings
array[0]='String one'     -- set each array item
array[1]='Another string'
array[2]='foobar'
loop i=0 to 2             -- display them
  say array[i]
  end
```

This example also shows NetRexx *line comments*; the sequence "--" (outside of literal strings or "/*" comments) indicates that the remainder of the line is not part of the program and is commentary.

B.10 Tracing

Defined as part of the language, NetRexx tracing often provides useful debugging information. The flow of execution of programs may be traced, and the execution trace can be viewed as it occurs or captured in a file. The trace can show each clause as it is executed, and optionally show the results of expressions, and so on. For example, the program:

```
trace results
number=1/7
parse number before '.' after
say after'.'before
```

would result in the trace:

```
2 *=*   number=1/7
  >v> number "0.142857143"
3 *=*   parse number before '.' after
  >v> before "0"
  >v> after "142857143"
4 *=*   say after'.'before
  >>> "142857143.0"
```

where the lines marked with "*=*" are the statements in the program, lines with ">v>" show results assigned to local variables, and lines with ">>>" show results of unnamed expressions.

B.11 Exception and Error Handling

NetRexx doesn't have a GOTO statement, but a SIGNAL statement is provided for abnormal transfer of control, such as when something unusual occurs. Using SIGNAL raises an *exception*; all control statements are then "unwound" until the exception is caught by a control statement that specifies a suitable CATCH statement for handling the exception.

Exceptions are also raised when various errors occur, such as attempting to divide a number by zero. For example:

```
say 'Please enter a number:'
number=ask
do
  say 'The reciprocal of' number 'is:' 1/number
catch Exception
  say 'Sorry, could not divide "'number'" into 1'
end
```

Here, the CATCH statement will catch any exception that is raised when the division is attempted (conversion error, divide by zero, and so on).

Any control statement that ends with END (DO, LOOP, or SELECT) may be modified with one or more CATCH statements to handle exceptions.

B.12 Things that aren′t Strings

In all the examples so far, the data being manipulated (numbers, words, and so on) is expressed as a string of characters. Many things, however, can be expressed more easily in some other way, so NetRexx allows variables to refer to other collections of data, which are known as *objects*.

Objects are defined by a name that lets NetRexx determine the data and methods that are associated with the object. This name identifies the type of the object, and is usually called the *class* of the object.

For example, an object of class Oblong might represent an oblong to be manipulated and displayed. The oblong could be defined by two values: its width and its height. These values are called the *properties* of the Oblong class.

Most methods associated with an object perform operations on the object; for example a “size” method might be provided to change the size of an Oblong object. Other methods are used to construct objects (just as for NetRexx arrays, an object must be constructed before it can be used). In NetRexx and Java, these *constructor* methods always have the same name as the class of object that they build (Oblong, in this case).

Here′s how an Oblong class might be written in NetRexx (by convention, this would be written in a file called Oblong.nrx; Java expects the name of the file to match the name of the class inside it):

```
/* Oblong.nrx -- simple oblong class */
class Oblong
  width       -- size (X dimension)
  height      -- size (Y dimension)

  /* Constructor method to make a new oblong */
  method Oblong(new_width, new_height)
    -- when we get here, a new (uninitialized) object has been
    -- created.  Copy the parameters we have been given to the
    -- properties of the object:
    width=new_width; height=new_height

  /* Change the size of an Oblong */
  method size(new_width, new_height) returns Oblong
    width=new_width; height=new_height
    return this   -- return the resized object

  /* Change the size of an Oblong, relative to its current size */
  method sizerelative(rel_width, rel_height) returns Oblong
    width=width+rel_width; height=height+rel_height
    return this

  /* 'Print' what we know about the oblong */
  method print
    say 'Oblong' width 'x' height
```

To summarize:

1. A class is started by the “class” statement, which names the class.

2. The class statement is followed by a list of the properties of the object. These can be assigned initial values, if required.

3. The properties are followed by the methods of the object. Each method is introduced by a method statement which names the method and describes the arguments that must be supplied to the method. The body of the method is ended by the next method statement (or by the end of the file).

The Oblong.nrx file is compiled just as any other NetRexx program, and should create a *class file* called Oblong.class. Here′s a program to try out the Oblong class:

```
/* tryOblong.nrx -- try the Oblong class */

first=Oblong(5,3)             -- make an oblong
first.print                   -- show it
first.sizerelative(1,1).print -- enlarge it and print it again

second=Oblong(1,2)            -- make another oblong
second.print                  -- and print it
```

When "tryOblong.nrx" is compiled, you′ll notice that the cross-reference listing of variables shows that the variables "first" and "second" have type "Oblong." These variables refer to Oblongs, just as the variables in earlier examples referred to REXX strings.

Once a variable has been assigned a type, it can only refer to objects of that type. This helps avoid errors where a variable refers to an object that it wasn′t meant to.

B.12.1 Programs are Classes

It′s worth pointing out here, that all the example programs in this document are in fact classes (you may have noticed that compiling them creates xxx.class files, where xxx is the name of the source file). The Java environment will allow a class to run as a stand-alone *application* if it has a constant method called "main" which takes an array of Java Strings as its argument.

If necessary (that is, if there is no class statement) NetRexx automatically adds the necessary class and method statement, and also a statement to convert the array of strings (each of which holds one word from the command string) to a single REXX string. The "toast" example could therefore have been written:

```
/* This wishes you the best of health. */
class toast
  method main(argwords=String[]) constant; arg=Rexx(argwords)
    say ′Cheers!′
```

B.13 Extending Classes

It′s common, when dealing with objects, to take an existing class and extend it. One way to do this is to modify the source code of the original class -- but this isn′t always available, and with many different people modifying a class, classes could rapidly get over-complicated.

Languages that deal with objects, such as NetRexx, therefore allow new classes of objects to be set up which are derived from existing classes. For example, if you wanted a different kind of Oblong in which the Oblong had a new property

that would be used when printing the Oblong as a rectangle, you might define it thus:

```
/* charOblong.nrx -- an oblong class with character */
class charOblong extends Oblong
  printchar      -- the character for display

  /* Constructor method to make a new oblong with character */
  method charOblong(new_width, new_height, new_printchar)
    super(new_width, new_height)      -- make an oblong
    printchar=new_printchar           -- and set the print character

  /* 'Print' the oblong */
  method print
    loop for super.height
      say printchar.copies(super.width)
      end
```

There are several things worth noting about this example:

1. The “extends Oblong” on the class statement means that this class is an extension of the Oblong class. The properties and methods of the Oblong class are *inherited* by this class (that is, appear as though they were part of this class).

   Another common way of saying this is that “charOblong” is a *subclass* of “Oblong” (and “Oblong” is the *superclass* of “charOblong”).

2. This class adds the “printchar” property to the properties already defined for Oblong.

3. The constructor for this class takes a width and height (just as with Oblong) and adds a third argument to specify a print character. It first invokes the constructor of its superclass (Oblong) to build an Oblong, and finally sets the printchar for the new object.

4. The new charOblong object also prints differently, as a rectangle of characters, according to its dimension. The “print” method (as it has the same name and arguments -- none -- as that of the superclass) replaces (overrides) the “print” method of Oblong.

5. The other methods of Oblong are not overridden, and therefore can be used on charOblong objects.

The charOblong.nrx file is compiled just as Oblong.nrx was, and should create a file called charOblong.class. Here's a program to try it out:

```
/* trycharOblong.nrx -- try the charOblong class */

first=charOblong(5,3,'#')       -- make an oblong
first.print                     -- show it
first.sizerelative(1,1).print   -- enlarge it and print it again

second=charOblong(1,2,'*')      -- make another oblong
second.print                    -- and print it
```

This should create the two charOblong objects, and print them out in a simple “character graphics” form. Note the use of the method “sizerelative” from Oblong to resize the charOblong object.

B.13.1 Optional Arguments

All methods in NetRexx may have optional arguments (omitted from the right) if desired. For an argument to be optional, you must supply a default value. For example, if the charOblong constructor was to have a default printchar value, its method statement could have been written:

```
method charOblong(new_width, new_height, new_printchar='X')
```

which indicates that if no third argument is supplied then "X" should be used. A program creating a charOblong could then simply write:

```
first=charOblong(5,3)       -- make an oblong
```

which would have the same effect as if "X" were specified as the third argument.

B.14 Binary Types and Conversions

The Java environment supports, and indeed requires, the notion of fixed-precision "primitive" binary types, which correspond closely to the binary operations usually available at the hardware level in computers. In brief, these types are:

byte, short, int, and long signed integers that will fit in 8, 16, 32, or 64 bits respectively

float and double signed floating point numbers that will fit in 32 or or 64 bits respectively.

char an unsigned 16-bit quantity, holding a Unicode character

boolean a 1-bit logical value, representing "false" or "true."

Objects of these types are handled specially by the environment "under the covers" in order to achieve maximum efficiency; in particular, they cannot be constructed like other objects -- their value is held directly. This distinction rarely matters to the NetRexx programmer: in the case of string literals an object is constructed automatically; in the case of an *int* literal, an object is not constructed.

Further, NetRexx automatically allows the conversion between the various forms of character strings in Java (String, char, char[], and REXX) and the primitive types listed above. The "golden rule" that is followed by NetRexx is that any automatic conversion which is applied must not lose information: either it can be determined at compile time that the conversion is safe (as in int -> String) or it will be detected at run time if the conversion fails (as in String -> int).

The automatic conversions greatly simplify the writing of programs for the Java environment: the exact type of numeric and string-like method arguments rarely needs to be a concern of the programmer.

For certain applications where early checking or performance override other considerations, NetRexx provides options for different treatment of the primitive types:

1. options strictassign -- ensures exact type matching for all assignments. No conversions (including those from shorter integers to longer ones) are applied. This option provides stricter type-checking than Java, and ensures that all types are an exact match.

2. options binary -- uses Java fixed precision arithmetic on binary types (also, literal numbers, for example, will be treated as binary, and local variables will be given “native” Java types such as *int* or *String*, where possible).

   Binary arithmetic currently gives better performance than Rexx decimal arithmetic, but places the burden of avoiding overflows and loss of information on the programmer.

The options statement (which may list more than one option) is placed before the first class statement in a file.

You may also explicitly assign a type to an expression or variable:

```
i=int 3000000    -- 'i' is an 'int' with initial value 3000000
j=int 4000000    -- 'j' is an 'int' with initial value 4000000
k=int            -- 'k' is assigned type 'int', with no initial value
say i*j          -- carry out multiplication and display the result
k=i*j            -- carry out multiplication and assign result to 'k'
```

This example also illustrates one difference between “options nobinary” and “options binary.” With the former (the NetRexx default) the SAY would display “1.20000000E+13” and a Conversion overflow would be reported when the same expression is assigned to the variable “k.”

With “options binary,” binary arithmetic would be used for the multiplications, and so no error would be detected; the SAY would display “-138625024” and the variable “k” takes the incorrect result.

B.14.1 Binary Types in Practice

In practice, explicit type assignment is only occasionally needed in NetRexx. Those conversions that are necessary for using existing classes (or those that use “options binary”) are generally automatic. For example, here is an "Applet" for use by Java-enabled browsers:

```
/* A simple graphics Applet */
class Rainbow extends Applet
  method paint(g=Graphics)                -- called to repaint the window
    maxx=size.width-1
    maxy=size.height-1
    loop y=0 to maxy
      col=Color.getHSBColor(y/maxy, 1, 1) -- select a colour
      g.setColor(col)                     -- set it
      g.drawLine(0, y, maxx, y)           -- and fill a slice
    end y
```

In this example, the variable “col” will have type “Color,” and the three arguments to the method “getHSBColor” will all automatically be converted to type “float.” As no overflows are possible in this particular example, “options binary” may be added to the top of the program with no other changes being necessary.

B.15 Summary and Information Sources

The NetRexx language, as you will have seen, allows the writing of programs for the Java environment with a minimum of overhead and “boilerplate syntax”; using NetRexx for writing Java classes could increase your productivity by 30% or more.

Further, by simplifying the variety of numeric and string types of Java down to a single class that follows the rules of REXX strings, programming is greatly simplified. Where necessary, however, full access to all Java types and classes is available.

Other examples are available, including both stand-alone applications and samples of applets for Java-enabled browsers (for example, an applet that plays an audio clip, and another that displays the time in English). You can find these from the NetRexx web pages, at URL

```
http://www2.hursley.ibm.com/netrexx/
```

Also at that location, you'll find a more in-depth treatment of the language, and downloadable packages containing the NetRexx software and documentation. The software should run on any platform that supports the Java Development Kit.

Appendix C. Special Notices

This publication is intended to help VM/ESA technical professionals and programmers to deploy Java and NetRexx applications using VM/ESA as a server platform. The information in this publication is not intended as the specification of any programming interfaces that are provided by Java, NetRexx, or VM/ESA Version 2 Release 3. See the PUBLICATIONS section of the IBM Programming Announcement for NetRexx, and for VM/ESA Version 2 Release 3, for more information about what publications are considered to be product documentation.

As this redbook is being written, the VM/ESA platform′s port of the Java Developer Kit, as developed to run with the OpenEdition Shell and Utilities feature, is proceeding through the Java Compatible test process. It will be made Generally Available ("GA") and will carry the Java Compatible logo once it is proven to pass the Java Compatible test suite.

An alternative execution environment, created by the VM platform developers in Endicott together with the authors of this redbook in Poughkeepsie, removes the need for customers to purchase the OpenEdition Shell and Utilities feature by offering a "shell-less" Java implementation. As this environment has not yet been verified as Java Compatible, it is offered as a "Beta" release only.

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, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

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

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

© Copyright IBM Corp. 1998

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.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

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.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

VM/ESA

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

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

Other company, product, and service names may be trademarks or service marks of others.

Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 165.

- *Creating Java Applications Using NetRexx*, SG24-2216
- *OpenEdition for VM/ESA Implementation and Administration Guide*, SG24-4747
- *Web Server Solutions for VM/ESA*, SG24-4874
- *S/390 - IBM Network Station - Getting Started*, SG24-4954

D.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

D.3 Other IBM Publications

These publications are also relevant as further information sources:

- *OpenEdition for VM/ESA User's Guide*, SC24-5727
- *OpenEdition for VM/ESA Command Reference*, SC24-5728
- *VM/ESA Planning and Administration*, SC24-5750
- *VM/ESA CMS File Pool Planning, Administration, and Operation*, SC24-5751
- *VM/ESA Connectivity Planning, Administration, and Operation*, SC24-5756.
- *TCP/IP Function Level 310 Planning and Customization*, SC24-5847.
- *PIPE Command Programming Interface*, available from the Pipelines Run Time Distribution web page at `http://pucc.princeton.edu/~pipeline/`

© Copyright IBM Corp. 1998

D.4 At Your Local Bookstore

We've found these textbooks to be useful during the development of this Redbook; the more stars (•), the more useful. These publications are available through your local book vendor:

- Cowlishaw, M. F., 1997. *The NetRexx Language*, • • • • ISBN 0-13-806332-X. Prentice Hall, Inc.
- Flanagan, D., 1997. *Java in a Nutshell (second edition)*, • • ISBN 1-56592-262- X. O'Reilly and Associates
- Lemay, L. and C. L. Perkins, 1997. *Teach Yourself Java 1.1 in 21 Days* • ISBN 1-57521-142-4. Sams.net Publishing
- Heller, P. and S. Roberts, 1997. *Java 1.1 Developer's Handbook*, • • • ISBN 0-7821-1919-0. SYBEX
- Deitel, H., and P. Deitel, 1998. *Java: How To Program (second edition)*, • ISBN 0-13-899394-7. Prentice Hall, Inc.
- Naughton, P., 1996. *The Java Handbook*, ISBN 0-07-882199-1. McGraw-Hill
- Gosling, J., ***et. al.***, 1996. *The Java Application Programming Interface, Vol. 1 (Core Packages)*, • • ISBN 0-201-63453-8. Addison-Wesley Publishing Company
- Gosling, J., B. Joy and G. Steele, 1996. *The Java Language Specification*, ISBN 0-201-63456-2. Addison-Wesley Publishing Company
- Arnold, K. and J. Gosling, 1996. *The Java Programming Language*, ISBN 0-201-63455-4. Addison-Wesley Publishing Company
- Lindholm, T. and F. Yellin, 1997. *The Java Virtual Machine Specification*, ISBN 0-201-63452-X Addison-Wesley Publishing Company

D.5 On the Web

The following are electronic publications, available on the World Wide Web at the URLs listed:

- *VM/ESA Java and NetRexx Home Page*

  `http://www.vm.ibm.com/java/`
- *NetRexx Home Page*

  `http://www2.hursley.ibm.com/netrexx/`
- *IBM's Java-related Technology*

  `http://ncc.hursley.ibm.com/javainfo/`
- *Sun Microsystems Java Technology Home Page*

  `http://www.javasoft.com/`
- *VM/ESA Support for IBM Network Stations*

  `http://www.vm.ibm.com/networkstation/`
- *IBM Network Station Technical Support and Services*

  `http://www.pc.ibm.com/networkstation/support/`

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com/.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Redbooks Web Site on the World Wide Web**

  http://w3.itso.ibm.com/

- **PUBORDER** — to order hardcopies in the United States
- **Tools Disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

- **REDBOOKS Category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

Redpieces

For information so current it is still in the process of being written, look at ″Redpieces″ on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.html). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

© Copyright IBM Corp. 1998

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

Index # 4421 Abstracts of new redbooks
Index # 4422 IBM redbooks
Index # 4420 Redbooks for last six months

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com/
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

Redpieces

For information so current it is still in the process of being written, look at ″Redpieces″ on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.html). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

A

abend. Abnormal end of task.

access mode. (1) A technique that is used to obtain a particular logical record from, or to place a particular logical record into, a file assigned to a mass storage device. (2) The manner in which files are referred to by a computer. Access can be sequential (records are referred to one after another in the order in which they appear on the file), access can be random (the individual records can be referred to in a nonsequential manner), or access can be dynamic (records can be accessed sequentially or randomly, depending on the form of the input/output request).

action bar. The area at the top of a window that contains choices that give a user access to actions available in that window.

addressing. (1) The assignment of addresses to the instructions of a program. (2) A means of identifying storage locations. (3) In data communication, the way in which a station selects the station to which it is to send data.

A-disk. In CMS, the primary user disk that is allocated to a CMS user. This read/write disk is used to store files created under CMS; such files are retained until deleted by the user. See also B-disk, CMS system disk, D-disk, virtual disk, Y-disk, Z-disk. Synonymous with primary user disk.

alias. An alternate label; for example, a label and one or more aliases may be used to refer to the same data element or point in a computer program.

allocate. (1) To assign a resource, such as a disk file, to perform a task. (2) A logical unit (LU) 6.2 application program interface (API) verb used to assign a session to a conversation for the conversation′s use. Contrast with deallocate.

application program. A program that is specific to the solution of an application problem. Synonymous with application software.

argument. (1) Any value of an independent variable; for example, a search key; a number identifying the location of an item in a table. (2) A parameter passed between a calling program and a called program.

array. (1) An arrangement of data in one or more dimensions: a list, a table, or a multidimensional arrangement of items. (2) In programming languages, an aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting.

assembler. A computer program that converts assembly language instructions into object code.

assignment. The process of giving values to variables.

attach. In programming, to create a task that can be executed asynchronously with the execution of the mainline code.

audio clip. In multimedia applications, a section of recorded audio material.

authorization. The process of granting a user either complete or restricted access to an object, resource, or function.

availability. (1) The ratio of the total time a functional unit is capable of being used to the total time the functional unit is required for use. (2) The degree to which a system or resource is ready when needed.

B

background. In multiprogramming, the conditions under which low-priority programs are executed.

backup copy. A copy of information or data that is kept in case the original is changed or destroyed.

batch. A group of records or data processing jobs brought together for processing or transmission.

binary file. A file that contains codes that are not part of the ASCII character set. Binary files can utilize all 256 possible values for each byte in the file.

block. (1) In programming languages, a compound statement that coincides with the scope of at least one of the declarations contained within it. A block may also specify storage allocation or segment programs for other purposes. (2) A string of data elements recorded or transmitted as a unit. The elements may be characters, words or physical records.

boilerplate. (1) A frequently used segment of stored text that may be combined with other text to create a new document. (2) In word processing and desktop publishing, text that is stored for repeated use in various documents; for example the wording of an edition notice.

boot. To prepare a computer system for operation by loading an operating system.

bridge. A functional unit that interconnects multiple LANs (locally or remotely) that use the same logical link control protocol but that can use different medium

© Copyright IBM Corp. 1998

access control protocols. A bridge forwards a frame to another bridge based on the medium access control (MAC) address.

built-in. In programming languages, pertaining to a language object that is declared by the definition of the programming language; for example, the built-in function SIN in PL/I, the predefined data type INTEGER in FORTRAN.

button. A graphical mechanism in a window that, when selected, results in an action; for example, a list button produces a list of choices.

byte. (1) A string that consists of a number of bits, treated as a unit, and representing a character. (2) A group of 8 adjacent binary digits that represent one EBCDIC character. (4) See n-bit byte.

C

cache. A special-purpose buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

calling program. A program that requests execution of another program (a called program).

cancel. To end a task before it is completed.

cascade. Deprecated term for overlap, or stack.

case-sensitive. Pertaining to the ability to distinguish between uppercase and lowercase letters.

catalog. A directory of files and libraries, with reference to their locations. A catalog may contain other information such as the types of devices in which the files are stored, passwords, and blocking factors.

character graphics. Graphics that are composed of symbols printed in a monospace font. Some symbols are stand-alone, others are intended for assembling larger figures.

character set. A finite set of different characters that is complete for a given purpose.

class method. In System Object Model, an action that can be performed on a class object. Synonymous with factory method.

clause. (1) In COBOL, an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry. See data clause, environment clause, file clause, report clause. (2) In SQL, a distinct part of a statement in the language structure, such as a SELECT clause or a WHERE clause.

client. A user or a functional unit that receives shared services from a server.

client-server. In TCP/IP, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server.

close. The function that ends the connection between a file and a program, and ends the processing. Contrast with open.

code page. (1) An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, assignment of characters and meanings to 128 code points for a 7-bit code. (2) In the Print Management Facility, a font library member that associates code points and character identifiers. A code page also identifies invalid code points. (3) In AFP support, a font file that associates code points and graphic character identifiers.

command. (1) A statement used to request a function of the system. A command consists of the command name abbreviation, which identifies the requested function, and its parameters. (2) In SDLC, a frame transmitted by a primary station. Asynchronous balanced mode stations send both commands and responses. Contrast with response. (3) A request from a terminal for the performance of an operation or the execution of a particular program.

command line. On a display screen, a display line usually at the bottom of the screen, in which only commands can be entered.

compact. Synonym for compress.

compatible. Pertaining to computers on which the same programs can be run without appreciable alteration.

compilation. Translation of a source program into an executable program (an object program).

compiler. A program that translates instructions written in a high-level programming language into machine language.

compiler options. Keywords that can be specified to control certain aspects of compilation. Compiler options can control the nature of the load module generated by the compiler, the types of printed output to be produced, the efficient use of the compiler, and the destination of error messages.

concatenate. To join two strings.

configuration file. A file that specifies the characteristics of a system or subsystem.

console. A part of a computer used for communication between the operator or maintenance engineer and the computer.

continuation line. A line of a source statement into which characters are entered when the source statement cannot be contained on the preceding line or lines.

control file. In CMS, the file that contains records that identify the updates to be applied and the macrolibraries, if any, needed to assemble that source program.

control statement. In programming languages, a statement that is used to alter the continuous sequential execution of statements; a control statement may be a conditional statement, such as IF, or an imperative statement, such as STOP.

control variable. In PL/I, a variable that is used to control the operation of a program, as in a DO statement.

conversion. (1) In programming languages, the transformation between values that represent the same data item but belong to different data types. Information may be lost due to conversion since accuracy of data representation varies among different data types. (2) The process of changing from one method of data processing to another or from one data processing system to another. (3) The process of changing from one form of representation to another; for example, to change from decimal representation to binary representation.

cross-reference listing. The portion of the compiler listing that contains information on where files, fields, and indicators are defined, referenced, and modified in a program.

cursor. A movable, visible mark used to indicate a position of interest on a display surface.

customization. The process of designing a data processing installation or network to meet the requirements of particular users.

D

data component. The part of a VSAM data set, alternate index, or catalog that contains the data records of an object.

data type. An attribute used for defining data as numeric or character.

database. (1) A collection of data with a given structure for accepting, storing, and providing, on demand, data for multiple users. (2) A collection of interrelated data organized according to a database schema to serve one or more applications.

debugging. Acting to detect and correct errors in software or system configuration.

decimal notation. A notation that uses ten different characters, usually the decimal digits; for example, the character string 196912312359, construed to represent the date and time one minute before the start of the year 1970. Classification (UDC).

default value. A value assumed when no value has been specified. Synonymous with assumed value.

delay. The amount of time by which an event is retarded.

deprecated. Pertaining to terms that should not be used.

desktop. A folder that fills the entire screen and holds all the objects with which the user can interact to perform operations on the system.

directory. (1) A table of identifiers and references to the corresponding items of data. (2) A type of file containing the names and controlling information for other files or other directories.

directory tree. An outline of all the directories and subdirectories on the current drive.

dispatcher. The program in an operating system that places jobs or tasks into execution.

distributed application. An application for which the component application programs are distributed between two or more interconnected processors.

download. To transfer programs or data from a computer to a connected device, typically a personal computer. Contrast with upload.

dummy. Pertaining to the characteristic of having the appearance of a specified thing but not having the capacity to function as such; for example, a dummy character, dummy plug, or dummy statement.

duplicate. To copy from a source to a destination that has the same physical form as the source; for example, to punch new punched cards with the same pattern of holes as an original punched card.

E

echo. In data communication, a reflected signal on a communications channel. On a communications terminal, each signal is displayed twice, once when entered at the local terminal and again when returned over the communications link. This allows the signals to be checked for accuracy.

encoding. The conversion of data to machine-readable format; the final step in the process of converting an analog signal into a digital signal.

The three steps are: sampling, quantizing, and encoding.

end of file (EOF). A coded character recorded on a data medium to indicate the end of the medium.

end user. (1) A person, device, program, or computer system that utilizes a computer network for the purpose of data processing and information exchange. (2) The ultimate source or destination of application data flowing through an SNA network. An end user can be an application program or a workstation operator.

entity. Any concrete or abstract thing of interest, including associations among things; for example, a person, object, event, or process that is of interest in the context under consideration, and about which data may be stored in a database.

environment variable. Any of a number of variables that describe the way an operating system is going to run and the devices it is going to recognize.

error message. An indication that an error has been detected.

escape. To return to the original level of a user interface.

event. An occurrence of significance to a task; for example, the completion of an asynchronous operation, such as an input/output operation.

executable program. A program that has been link-edited and therefore can be run in a processor.

execution time. The amount of time needed for the execution of a particular computer program.

exit. To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on.

expand. To return compressed data to their original form.

expression. In programming languages, a language construct for computing a value from one or more operands; for example, literals, identifiers, array references, and function calls.

external routine. In REXX, a program external to the user′s program, language processor, or both. These routines can be written in any language, including REXX that supports the system-dependent interfaces used by REXX to start it.

extract. (1) To select and remove from a group of items those items that meet a specific criteria. (2) To obtain; for example, information from a file.

F

factorial. The product of the positive integers 1, 2, 3, up to and including a given integer.

feature. A part of an IBM product that may be ordered separately by the customer.

fetch. In virtual storage systems, to bring load modules or program phases from auxiliary storage into virtual storage.

file description. A part of a file where file and field attributes are described.

file name. (1) A name assigned or declared for a file. (2) The name used by a program to identify a file.

file server. A high-capacity disk storage device or a computer that each computer on a network can use to access and retrieve files that can be shared among the attached computers.

file system. In the AIX operating system, the collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk. See distributed file system, virtual file system.

file tree. In the AIX operating system, the complete directory and file structure of a particular node, starting at the root directory. A file tree contains all local and remote mounts performed on directories and files.

fill. (1) In a token-ring network, a specified bit pattern that a transmitting data station sends before or after transmission frames, tokens, or abort sequences to avoid what would otherwise be interpreted as an inactive or indeterminate transmitter state. (2) In computer graphics, a designated area of the screen that is flooded with a particular color.

flag. A variable indicating that a certain condition holds.

font. A family of characters of a given size and style; for example, 9-point Helvetica.

frame. A data structure that consists of fields, predetermined by a protocol, for the transmission of user data and control data. The composition of a frame, especially the number and types of fields, may vary according to the type of protocol. Synonymous with transmission frame.

function call. An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by the called function. A function call contains the name of the function to which control moves and a parenthesized list of values.

G

generate. (1) To produce a computer program by selecting subsets from skeletal code under the control of parameters. (2) To produce assembler language statements from the model statements of a macrodefinition when the definition is called by a macroinstruction.

global. In programming languages, pertaining to the relationship between a language object and a block in which the language object has a scope extending beyond that block but contained within an encompassing block.

graphical user interface (GUI). A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

group ID (GID). In RACF, a string of one to eight characters that identifies a group. The first character must be A through Z, #, $, or @. The rest can be A through Z, #, $, @, or 0 through 9. (2) In the AIX operating system, a number that corresponds to a specific group name. The group ID can often be substituted in commands that take a group name as a value.

group name. (1) A generic name for a collection of I/O devices; for example, DISK or TAPE. See also device type, unit address. (2) In the AIX operating system, a name that uniquely identifies a group of users to the system and that contains one to eight alphanumeric characters, beginning with an alphabetic, #, $, or > character. (3) In RACF, one to eight alphanumeric characters beginning with an alphabetic, #, $, or > character that identify a group.

H

hardware. (1) All or part of the physical components of an information processing system, such as computers or peripheral devices. (2) The equipment, as opposed to the programming, of a system.

hexadecimal. Pertaining to a system of numbers to the base 16; hexadecimal digits range from 0 through 9 and A through F, where A represents 10 and F represents 15.

home directory. In the AIX operating system: (a) A directory associated with an individual user. (b) The user's current directory after login or after issuing the cd command with no argument. (c) A parameter that supplies the full path name of the home directory for the transaction program.

host. In TCP/IP, any system that has at least one Internet address associated with it. A host with multiple network interfaces may have multiple Internet addresses associated with it.

I

icon. A graphic symbol, displayed on a screen, that a user can point to with a device such as a mouse in order to select a particular function or software application. Synonymous with pictogram.

implementation. The system development phase at the end of which the hardware, software and procedures of the system considered become operational.

incremental backup. In the Data Facility Hierarchical Storage Manager, the process of copying data sets that have been opened for reasons other than read-only access since the last backup version was created and that meet the backup frequency criteria.

informational message. A message that provides information but does not require a response.

initial value. A value assumed until explicitly changed.

insert. To introduce data between previously stored items of data.

installation script. In the AIX operating system, a shell procedure or executable file created by the developer of an application program to install the program. The script file must follow specific guidelines in order to be compatible with the program installation tools that are provided in the operating system.

instruction set. The set of instructions of a computer, of a programming language, or of the programming languages in a programming system.

integer. A positive or negative whole number, that is, an optional sign followed by a number that does not contain a decimal place or zero.

interaction. A basic unit used to record system activity, consisting of the acceptance of a line of terminal input, processing of the line, and a response, if any.

interface. A shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions.

interleave. To arrange parts of one sequence of things or events so that they alternate with parts of one or more other sequences of the same nature and so that each sequence retains its identity.

internet. A collection of packet-switching networks that are physically interconnected by Internet Protocol (IP) gateways. These networks use protocols that allow them to function as a large, composite network.

Internet. A wide area network connecting thousands of disparate networks in industry, education, government, and research. The Internet network uses TCP/IP as the standard for transmitting information.

interpreter. A program that translates and executes each instruction of a high-level programming language before it translates and executes the next instruction.

invoke. To start a command, procedure, or program.

K

kernel. The part of an operating system that performs basic functions such as allocating hardware resources.

keyword. In programming languages, a lexical unit that, in certain contexts, characterizes some language construct; for example, in some contexts, IF characterizes an if-statement. A keyword normally has the form of an identifier.

L

label. (1) In programming languages, a language construction naming a statement and including an identifier. (2) An identifier within or attached to a set of data elements.

library. A named area on disk that can contain programs and related information (not files). A library consists of different sections, called library members.

limited interface. In the AIX operating system, a set of system calls that provides a limited function interface.

line. (1) The portion of a data circuit external to data circuit-terminating equipment (DCE), that connects the DCE to a data switching exchange (DSE), that connects a DCE to one or more other DCEs, or that connects a DSE to another DSE. (2) On a terminal, one or more characters entered before a return to the first printing or display position.

line mode. (1) In VM, when using the System Product Editor or the CMS Editor, the mode of operation of a display terminal that is equivalent to using a typewriter terminal; that is, the terminal displays a chronological log of the XEDIT or EDIT subcommands entered, the lines affected by the editing (unless it is suppressed), and the system responses. Contrast with display mode. (2) A form of screen presentation in which the information is presented a line at a time in the message area of the terminal screen. Contrast with full-screen mode.

list box. A control that contains scrollable choices from which a user can select one choice.

literal string. A string that does not contain pattern-matching characters and can therefore be interpreted just as it is. Contrast with regular expression.

load. To bring all or part of a computer program into memory from auxiliary storage so that the computer can run the program.

local variable. A variable that is defined and used only in one specified portion of a computer program.

log. (1) To record; for example, to log all messages on the system printer. (2) Synonym for journal.

logical. Pertaining to a view or description of data that does not depend on the characteristics of the computer system or of the physical storage. Contrast with physical.

long string. In SQL, a string whose actual length, or a varying-length string whose maximum length, is greater than 254 bytes or 127 double-byte characters.

looping. Repetitive execution of the same statement or statements, usually controlled by a DO statement.

M

macro. Synonym for macroinstruction.

mainline program. A program that performs primary functions, passing control to routines and subroutines for the performance of more specific functions.

mainframe. A computer, usually in a computer center, with extensive capabilities and resources to which other computers may be connected so that they can share facilities.

maintenance. Any activity intended to retain a functional unit in, or to restore it to, a state in which it can perform its required function. Maintenance includes keeping a functional unit in a specified state by performing activities such as tests, measurements, replacements, adjustments, and repairs.

map. To establish a set of values having a defined correspondence with the quantities or values of another set; for example, to evaluate a mathematical function, that is, to establish the values of the dependent variable for values of the independent variable or variables of immediate concern.

matching. The technique of comparing the keys of two or more records to select items for a particular stage of processing or to reject invalid records.

memory. All of the addressable storage space in a processing unit and other internal storages that is used to execute instructions. Synonymous with main storage.

menu. A list of options displayed to the user by a data processing system, from which the user can select an action to be initiated.

menu bar. (1) In the AIX operating system, a rectangular area at the top of the client area of a window that contains the titles of the standard pull-down menus for that application. See also scroll bar. (2) The area near the top of a window, below the title bar and above the rest of the window, that contains choices that provide access to other menus.

minidisk. Synonym for virtual disk.

model. The conceptual and operational understanding that a person has about something.

modification. (1) An addition or change to stored data or a deletion of stored data. (2) The change or customization of a system, subsystem, or application to work more effectively at a given installation.

module. A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine.

monitor. Software or hardware that observes, supervises, controls, or verifies operations of a system.

mount. (1) To place a data medium in a position to operate. (2) To make recording media accessible.

multitasking. A mode of operation that provides for concurrent performance, or interleaved execution of two or more tasks.

multithreading. Pertaining to concurrent operation of more than one path of execution within a computer.

N

native. Deprecated term for IBM-supplied, basic, required, or stand-alone.

network administrator. A person who manages the use and maintenance of a network.

networking. (1) In a multiple-domain network, communication between domains. Synonymous with cross-domain communication. See extended networking. (2) Loosely, making use of the services of a network.

notation. (1) A set of symbols and the rules for their use for the representation of data. (2) A system of characters, symbols, or abbreviated expressions used to express technical facts or qualities.

nucleus. That part of a control program resident in main storage. Synonymous with resident control program.

null string. A string containing no element.

numeric data. (1) Data represented by numerals. (2) Data in the form of numerals and some special characters; for example, a date represented as 81/01/01.

O

object. (1) In computer security, anything to which access is controlled; for example, a file, a program, an area of main storage. (2) A passive entity that contains or receives data; for example, bytes, blocks, clocks, fields, files, directories, displays, keyboards, network nodes, pages, printers, processors, programs, records, segments, words. Access to an object implies access to the information it contains. (3) Something that a user works with to perform a task. Text and graphics are examples of objects.

offset. (1) The number of measuring units from an arbitrary starting point in a record, area, or control block, to some other point. (2) The distance from the beginning of an object to the beginning of a particular field.

online. (1) Pertaining to the operation of a functional unit when under the direct control of the computer. (2) Pertaining to a user′s ability to interact with a computer.

open. The function that connects a file to a program for processing.

operand. An entity on which an operation is performed.

operating system (OS). Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible.

operator. A symbol that represents an operation to be done.

option. A specification in a statement that may be used to influence the execution of the statement.

output file. (1) A file that contains the results of processing. (2) A file that has been opened in order to allow records to be written.

overhead. In a computer system, the time, operations, and resources used for operating system functions, rather than for application programs.

override. (1) A parameter or value that replaces a previous parameter or value. (2) The attributes specified at run time that change the attributes specified in the file description or in the program.

P

padding. Concatenating a string with one or more characters, called fillers, usually in order to achieve a specific length of the string.

panel. (1) A set of logically related information displayed on the screen for the purpose of communicating information to or from a computer user. (2) A formatted display of information that appears on a display screen.

parameter. (1) A variable used in conjunction with a command to affect its result. (2) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. (3) Data passed between programs or procedures.

parent directory. (1) In VM, the directory for a CMS disk that has a disk extension defined for it via the ACCESS command. (2) The directory one level above the current directory.

parse. (1) In systems with time sharing, to analyze the operands entered with a command and create a parameter list for the command processor from the information. (2) In REXX, to split a string into parts, using function calls or by using a parsing template on the ARG, PARSE, or PULL instructions.

password. (1) A value used in authentication or a value used to establish membership in a set of people having specific privileges. (2) A unique string of characters known to a computer system and to a user, who must specify the character string to gain access to a system and to the information stored within it. (3) In computer security, a string of characters known to the computer system and a user, who must specify it to gain full or limited access to a system and to the data stored within it.

paste. To copy a page, object, or picture into an existing folder or picture.

path. (1) The route used to locate files; the storage location of a file. A fully qualified path lists the drive identifier, directory name, subdirectory name (if any), and file name with the associated extension. (2) In a network, any route between any two nodes. A path may include more than one branch. (3) The series of transport network components (path control and data link control) that are traversed by the information exchanged between two network accessible units.

pattern matching. The identifying of one of a predetermined set of items which has the closest resemblance to a given object, by comparing its coded representation against the representations of all the items.

peer. In network architecture, any functional unit that is in the same layer as another entity.

pipe. (1) To direct data so that the output from one process becomes the input to another process. The standard output of one command can be connected to the standard input of another with the pipe operator ( ]). Two commands connected in this way constitute a pipeline. (2) A one- way communication path between a sending process and a receiving process.

pipeline. (1) A serial arrangement of processors or a serial arrangement of registers within a processor. Each processor or register performs part of a task and passes results to the next processor; several parts of different tasks can be performed at the same time. (2) A direct, one- way connection between two or more processes.

platform. The operating system environment in which a program runs.

pointer. (1) A data element that indicates the location of another data element.

port. (1) An access point for data entry or exit. (2) A connector on a device to which cables for other devices such as display stations and printers are attached. Synonymous with socket.

portability. (1) The capability of a program to be executed on various types of data processing systems without converting it to a different language and with little or no modification. (2) The ability to run a program on more than one computer without modifying it.

precision. A measure of the ability to distinguish between nearly equal values; for example, four-place numerals are less precise than six-place numerals; nevertheless, a properly computed four-place numeral may be more accurate than an improperly computed six-place numeral.

predefined. Synonym for built-in.

procedure. (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a procedure call. (2) A set of related control statements that cause one or more programs to be performed.

process. (1) A course of the events defined by its purpose or by its effect, achieved under given

conditions. (2) In data processing, the course of events that occurs during the execution of all or part of a program.

processor. In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit.

profile. (1) Data that describes the significant characteristics of a user, a group of users, or one or more computer resources. (2) In computer security, a description of the characteristics of an entity to which access is controlled.

programmer. A person who designs, writes, and tests computer programs.

programming environment. An integrated collection of software and hardware to support the development of computer programs.

programming language. An artificial language for expressing computer programs.

prompt. (1) A visual or audible message sent by a program to request the user's response. (2) A displayed symbol or message that requests input from the user or gives operational information; for example, on the display screen of an IBM personal computer, the DOS A> prompt. The user must respond to the prompt in order to proceed.

property. A unique characteristic of an object that can be changed or modified. The properties of an object describe the object. Type style is an example of a property.

protocol. (1) A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (2) In Open Systems Interconnection architecture, a set of semantic and syntactic rules that determine the behavior of entities in the same layer in performing communication functions. (3) In SNA, the meanings of and the sequencing rules for requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

prototype. A model or preliminary implementation suitable for evaluation of system design, performance, and production potential, or for better understanding or determination of the requirements.

push button. A rectangle with text inside. Push buttons are used in windows for actions that occur immediately when the push button is selected.

Q

query. (1) A request for data from a database, based on specified conditions; for example, a request for availability of a seat on a flight reservation system. (2) In interactive systems, an operation at a terminal that elicits a response from the system. (3) A request for information from a file based on specific conditions; for example, a request for a list of all customers whose balance is greater than $1000.

quick start. Synonym for system restart.

quit. A key, command, or action that tells a system to return to a previous state or stop a process.

R

range. The set of values that a quantity or function may take.

reader. (1) A part of an operating system scheduler that reads an input stream into the system. (2) A program that reads jobs from an input device or database file and places them on a job queue. (3) In RJE, a program that reads jobs from a database file or interactive display station and sends them to the host system.

real name. The name by which a resource is identified in its native network.

receive. To obtain and store data.

record. A set of one or more related data items grouped for processing.

record format. The definition of how data are structured in the records contained in a file. The definition includes record name, field names, and field descriptions, such as length and data type. The record formats used in a file are contained in the file description.

recover. After an execution failure, to establish a previous or new status from which execution can be resumed.

regenerate. To restore information to its original state.

relative path name. In the AIX operating system, the name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

remainder. In a division operation, the number or quantity that is the undivided part of the dividend, having an absolute value less than the absolute value of the divisor, and that is one of the results of a division operation.

remote host. Any host on a network except the one at which a particular operator is working. Synonymous with foreign host.

remove. (1) In journaling, to remove the after-images of records from a physical file member. The file then contains the before-images of the records in the journal. Contrast with apply. (2) In the IBM Token-Ring Network, to take an attaching device off the ring.

reproduce. Synonym for duplicate.

residue. In computer security, data remaining in a data medium but not associated with a data object.

resource. Any facility of a computing system or operating system required by a job or task, and including main storage, input/output devices, processing unit, data sets, and control or processing programs.

restore. To return to an original value or image; for example, to restore data in main storage from auxiliary storage.

retrieve. To locate data in storage and read it so that it can be processed, printed, or displayed. Contrast with store.

return code. (1) A code used to influence the execution of succeeding instructions. (2) A value returned to a program to indicate the results of an operation requested by that program.

reusable. The attribute of a routine that allows the same copy of the routine to be used by two or more tasks.

root directory. The directory that contains all other directories in the system. See effective root directory.

routine. A program, or part of a program, that may have some general or frequent use.

run time. Synonym for execution time.

S

scan. To examine sequentially, part by part.

schedule. To select jobs or tasks that are to be dispatched. In some operating systems, other units of work such as input/output operations may also be scheduled.

scope. (1) The portion of an expression to which the operator is applied. (2) The portion of a computer program within which the definition of the variable remains unchanged.

scroll. To move a display image vertically or horizontally to view data that otherwise cannot be observed within the boundaries of the display screen.

scroll bar. A part of a window, associated with a scrollable area, that a user interacts with to see information that is not currently visible.

search time. The time interval required for the read/write head of a direct access storage device to locate a particular record on a track corresponding to a given address or key. Synonymous with rotational delay.

seek. To selectively position the access mechanism of a direct access device.

semantics. The relationships of characters or groups of characters to their meanings, independent of the manner of their interpretation and use.

send. In systems with VTAM, to place a message on a line for transmission from the computer to a terminal. Contrast with receive.

server. (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (2) In a network, a data station that provides facilities to other stations; for example, a file server, a print server, a mail server.

service access point (SAP). In Open Systems Interconnection architecture, the point at which the services of a layer are provided by an entity of that layer to an entity of the next higher layer.

service virtual machine. In VM, a virtual machine that provides system services. These services include accounting, error recording, and services provided by licensed programs.

session. (1) In network architecture, for the purpose of data communication between functional units, all the activities which take place during the establishment, maintenance, and release of the connection. (2) The period of time during which a user of a terminal can communicate with an interactive system, usually, elapsed time between logon and logoff.

shared. Pertaining to the availability of a resource for more than one use at the same time.

shell. A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens and communicate them to the operating system. Shells simplify user interactions by eliminating the user's concern with operating system requirements. A computer may have several layers of shells for various levels of user interaction.

shutdown. The process of ending operation of a system or a subsystem, following a defined procedure.

simulate. To represent certain features of the behavior of a physical or abstract system by the behavior of another system; for example, to represent a physical phenomenon by means of operations performed by a computer or to represent the operations of a computer by those of another computer.

socket. The abstraction provided by the University of California′s Berkeley Software Distribution (commonly called Berkeley UNIX or BSD UNIX) that serves as an endpoint for communication between processes or applications.

software. (1) All or part of the programs, procedures, rules, and associated documentation of a data processing system. Software is an intellectual creation that is independent of the medium on which it is recorded. (2) Contrast with hardware.

source code. The input to a compiler or assembler, written in a source language. Contrast with object code.

source file. A file that contains source statements for such items as high-level language programs and data description specifications.

special file. In the AIX operating system, a file that provides an interface to input/output devices. There is at least one special file for each device connected to the computer. Contrast with directory.

stand-alone. Pertaining to operation that is independent of any other device, program, or system.

statement. In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations.

static. In programming languages, pertaining to properties that can be established before execution of a program; for example, the length of a fixed length variable is static.

storage allocation. The assignment of storage areas to specified data.

storage group (SG). A named collection of physical devices to be managed as a single object storage area. It consists of an object directory (DB2 table space), and object storage on DASD (DB2 table spaces) with optional library-resident optical volumes.

store. To place data into a storage device.

string. A sequence of elements of the same nature, such as characters considered as a whole.

structure. A variable that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types.

subclass. A class that is derived from another class. The subclass inherits the data and methods of the parent class and can define new methods or override existing methods to define new behavior not inherited from the parent class.

subroutine. (1) A sequence of instructions whose execution is invoked by a call. (2) A sequenced set of instructions or statements that may be used in one or more computer programs and at one or more points in a computer program. (3) A group of instructions that can be part of another routine or can be called by another program or routine.

subset. A set each element of which is an element of a specified other set.

substring. A part of a character string.

superclass. In the AIXwindows program and Enhanced X-Windows, a class of widgets that passes inheritable resources down the hierarchy to a lower subclass.

superuser authority. In the AIX operating system, the unrestricted authority to access and modify any part of the operating system, usually associated with the user who manages the system.

symbolic name. In a programming language, a unique name used to represent an entity such as a field, file, data structure, or label.

syntax. The rules governing the structure of a language.

syntax error. A compile-time error caused by incorrect syntax. See also semantic error.

system. In data processing, a collection of people, machines, and methods organized to accomplish a set of specific functions.

system profile. A file containing the default values used in system operations.

system resources. Those resources controlled by the system, such as programs, devices, and storage areas that are assigned for use in jobs.

T

target. (1) Pertaining to a storage device to which information is written. (2) The program or system to which a request is sent. (3) The location to which the information is destined.

template. A pattern to help the user identify the location of keys on a keyboard, functions assigned to

keys on a keyboard, or switches and lights on a control panel.

terminate. (1) In SNA products, a request unit that is sent by a logical unit (LU) to its system services control point (SSCP) to cause the SSCP to start a procedure for ending one or more designated LU-LU sessions. (2) To stop the operation of a system or device. (3) To stop execution of a program.

token. In a local area network, the symbol of authority passed successively from one data station to another to indicate the station temporarily in control of the transmission medium. Each data station has an opportunity to acquire and use the token to control the medium. A token is a particular message or bit pattern that signifies permission to transmit.

tool. Software that permits the development of an application program without using a traditional programming language.

trace. A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed.

transfer. To send data from one place and receive the data at another place.

tutorial. Information presented in a teaching format.

U

update. (1) To add, change, or delete items. (2) To modify a master file with current information according to a specified procedure.

uppercase. Pertaining to the capital letters, as distinguished from the small letters; for example, A, B, G, rather than a, b, g.

user ID. User identification.

user interface. Hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

user name. In RACF, one to twenty alphanumeric characters that represent a RACF-defined user.

user program. A user-written program.

user-friendly. Pertaining to the ease and convenience of use by humans.

utility program. A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program. Synonymous with service program.

V

validation. The checking of data for correctness or for compliance with applicable standards, rules, and conventions.

value set. A group of choices, usually graphical, from which a user can select one.

variable. In programming languages, a language object that may take different values, one at a time. The values of a variable are usually restricted to a certain data type.

verify. (1) To determine whether a transcription of data or other operation has been accomplished accurately. (2) To confirm correctness of something.

version. A separate IBM-licensed program, based on an existing IBM-licensed program, that usually has significant new code or new function. Each version has its own license, terms, conditions, product type number, monthly charge, documentation, test allowance (if applicable), and programming support category.

virtual console. In VM, a console simulated by CP on a terminal such as a 3270. The virtual device type and I/O address are defined in the VM directory entry for that virtual machine.

virtual machine (VM). A virtual data processing system that appears to be at the exclusive disposal of a particular user, but whose functions are accomplished by sharing the resources of a real data processing system.

virtual storage. The storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations.

W

window. (1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (2) An area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen. (3) A division of a screen in which one of several programs being executed concurrently can display information.

working directory. Synonym for current directory.

workstation. A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

write access. In computer security, permission to write to an object.

Y

Y-disk. An extension of the CMS system disk.

Z

zero. In data processing, the number that, when added to or subtracted from any other number, does not alter the value of the other number. Zero may have different representations in computers, such as positively or negatively signed zero (which may result from subtracting a signed number from itself) and floating-point zero (in which the fixed point part is zero while the exponent in the floating-point representation may vary).

List of Abbreviations

ACCT	ACCounT
AD	Application Development
ADM	Application Development Manager
ANSI	American National Standards Institute
API	Application Program Interface
ASCII	American national Standard Code for Information Interchange
BASIC	Beginners All-purpose Symbolic Instruction Code
BFS	Byte File System
BG	BackGround
BOOTP	BOOT Protocol
CET	Central European Time
CMS	Conversational Monitor System
CP	Control Program
CPU	Central Processing Unit
CRR	Coordinated Resource Recovery
CSL	Callable Services Library
DASD	Direct Access Storage Device
DCE	Distributed Computing Environment
DCSS	DisContiguous Shared Segment
DDR	Dasd Dump Restore
DLL	Dynamic Link Library
EBCDIC	Extended Binary Coded Decimal Interchange Code
EOF	End Of File
FTP	File Transfer Program
GDDM	Graphical Data Display Manager
GID	Group IDentifier
GMT	Greenwich Mean Time
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
I/O	Input/Output
IBM	International Business Machines
ID	IDentification/IDentifier
IP	Internet Protocol
IPL	Initial Program Load
ITSO	International Technical Support Organization
IUCV	Inter-User Communication Vehicle
LE/370	Linkage Environmet/370
NFS	Network File System
NLS	National Language Support
OS/2	Operating System/2
PC	Personal Computer
PL/I	Programming Language/1
POSIX	Portable Operating System Interface for Computer Environments
PTF	Program Temporary Fix
R/O	Read/Only
R/W	Read/Write
REL	RELease
REXX	REstructured eXtended eXecutor language
RS/6000	IBM RISC System/6000
RSCS	Remote Spooling Communications Subsystem
S/390	IBM System/390
SFS	Shared File System
SPOOL	Simultaneous Peripheral Operation On-Line
TAR	Tape ARchive
TCP/IP	Transmission Control Protocol/Internet Protocol
TFTP	Trivial File Transfer Protocol
UID	User IDentification
URL	Uniform Resource Locator
VM	Virtual Machine
VM/ESA	Virtual Machine/Enterprise Systems Architecture
VMLIB	VM LIBrary
VMNFS	Virtual Machine Network File System
VTAM	Virtual Telecommunications Access Method
XA	Extended Architecture
XEDIT	eXtended EDITor

© Copyright IBM Corp. 1998

Index

Special Characters

'/../VMBFS:VMSYS:ROOT/' 23
-classpath, Java option 33
-ms, Java option 143
-mx, Java option 143
_JAVA, file space 15
_NETREXX, file space 15
.profile 23, 32
$BFSEXEC XEDIT 26, 27, 29
$BFSLIST XEDIT 27

Numerics

8859_1, InputStreamReader 87
8859_1, OutputStreamReader 87

A

abbrev method, usage example 44
abbrev, usage example 47
abbreviations 183
AboutFrame
  class definition 60
  class definition and properties 63
  constructor method 64
  event classes 67, 68
  how to use 71
  main method 71
  other methods 66
  picture 55
  program interface 56
  program overview 61
  properties 63
  reusable class 55
absolute path 8, 9
acronyms 183
actionPerformed, usage example 68
addActionListener, usage example 68
addItemListener, usage example 68
ADDRESS, instruction 45
addressing the host 45
addWindowListener, usage example 67
adjustmentValueChanged(), usage example 112
aliases, BFS 14
aliases, SFS 14
ARG, parse 44
arg, REXX variable 52
arguments method 59, 60
arguments, method 158
array
  adding an element 66
  creation example 65
  declare example 63
  declaring 51
array *(continued)*
  fixed size 51
  getting all elements 51
  NetRexx 153
  string, using 53
  using 53
  variables 50
ASCII 142
  editing 26
  issues 81, 87, 94
  printLine 87
  readLine 87
  socket 87
  XEDIT 26
ASCXED EXEC 26
ask instruction, usage example 44
ASK, get keyboard input 44
attribute, object 57
authors, displaying 56
avoiding, empty frames 73
AWT 39

B

BFS
  directory structures 13
  file space 14
  links 14
  listing contents 22, 23, 24
  objects, commands 26
  objects, sorting 26
  reading 79
  writing 79
BFS directory
  current 31
  deleting 26
  renaming 26
  tree listing 27
  tree view 26
BFS file
  copying 26
  copying to CMS 26
  deleting 26
  editing 26
  executing commands 24
  renaming 26
  XEDIT 26
BFSLIST 22, 24
  calling from POSIX Shell 32
  EXEC 27
  installing 27
  shortcomings 27
  tailoring 27
BFSSTATE, PIPE 26

© Copyright IBM Corp. 1998

BFSTREE 26, 27
BFSTREE EXEC 29
bibliography 163
binary
  data types 159
  option 150, 159
  reading 79
  types 158
  writing 79
brackets, entering from keyboard 50
byte stream 83

C

C environment variables, setting 33
calls, function 40, 45
canceling running programs 77
case
  environment variable 34
  insensitive 39, 42
  mixed 34
  sensitive 22, 38, 42
catch instruction 42
cent sign, entering 77
CENV 34
character stream 83
checking, method parameters 49
class
  AboutFrame 60
  defined 57
  extend 91
  extending 156
  extends option 63
  files, introduction 38, 39
  instance 57
  instruction 62
  introduced 40
  Java class 42
  NetRexx 155
  private 58
  public 58
  REXX 41
  super 66
  system 72
  variables 60
  Vector 52
classfile 31
classpath 29, 31, 34, 142
CLASSPATH, environment variable 33
closing frame 67
closing window 67
CMS
  commands 93, 94
  pipeline, executing 45
  pipelines 37
  pipelines, obtaining variables 56
  running pipelines 96
  running REXX execs 95
  system profile 35
CMS *(continued)*
  util package 94
  writing 78
CMSPipe
  class 97
  fitting 96
  usage example 115
  Usage Notes 98
co-pipelines 96
codepage 81
  1047 84
  8859_1 84
  getting 85
  introduced 82
color, usage example 69
commands
  dir 32
  export 33
  for BFS files 24
  for BFS objects 26
comments, NetRexx 40
comments, REXX 40
comparing
  REXX with NetRexx 37
  REXX with NetRexx, instructions 42
  strings 42
compile
  Java 38
  Java program 26, 31
  Just In Time 39
  NetRexx 29, 30, 38, 39
  NetRexx program 26
  options, NetRexx 30
  REXX 38
  virtual storage 29
compound strings 50
  NetRexx 50
compound variable 50, 152
  nullpointerexception 141
  usage example 118
console
  control keys 77
  control sequences 77
  CTRL keys 77
  CTRL sequences 77
  reading 77
constant property 60
constructor
  defined 157
  method 58, 62
  method, usage example 64
continuation line 40
control keys 77
control sequences 77
converting, data type 41
copying a BFS file to CMS 26
copying BFS file 26

CP directory 23
CPU loop 145
creating
  a directory 11
  a link 11, 26
  an alias 11
  BFS aliases 14
  external links 14
  SFS aliases 14
  symbolic links 14
CTRL keys 77
CTRL sequences 77
current directory 31

D

data type 39, 40
  binary 159
  conversions 158
  converting 41
  mismatches 142
  native 41
  NetRexx 41
  REXX 41
declaring an array 51
declaring variables, NetRexx 41
defining working directory 22
deleting BFS directory 26
deleting BFS file 26
dir, POSIX Shell command 32
directory
  cp 23
  current 31
  listing contents 22, 23, 24
  parent 26
  structures, BFS 13
  tree 26
DIRLIST 13, 24
displaying authors 56
do instruction 42, 43
do label option 43

E

EBCDIC 142
  issues 81, 87, 94
  printLine 87
  readLine 87
editing ASCII file 26
editing BFS file 26
elements, getting all of an array 51
empty frames, avoiding 73
empty variable 41
enrolling
  a BFS user 11, 14
  an SFS user 11, 13
entering square brackets 50
environment variables
  CLASSPATH 33
environment variables *(continued)*
  PATH 33
  setting 33
  setting from CMS 34
erasing a directory 11
Error Messages 142
exception and error handling, NetRexx 154
exec() 93
EXECLOAD 47
executing
  CMS pipeline 45
  host commands 45
  NetRexx program 26
exit instruction 49
exit method 49
export, classpath 35
export, POSIX Shell command 33
expose instruction 56
exposing variables 56, 60
extend, class 91
extending classes 156
extends, usage example 63
external
  links 14, 25
  MOUNT 26
  subroutines 47

F

file not found 142, 143
file space
  _JAVA 15
  _NETREXX 15
  BFS 14
  SFS 13
FILELIST 22
filename 25
filetype 25
finally instruction 42
fitting
  *<java 97, 98
  *>java 96, 97, 98
  CMSPipe 96
  stage 96
fixed size arrays 51
frame
  avoiding empty 73
  closing 67
  handling events 67
  layout 65
  positioning on the screen 65
  sizing 65
frequently asked questions 141
FSROOT 23
FTP with BFS 110
FTP with SFS 109
function calls 40, 45
  nested 46
  NetRexx 46

function calls *(continued)*
REXX 45
functions, user defined 47

G

GETDATA REXX 104, 111
GetEncoding, getting codepage 85
getProperties(), usage example 84
getProperty, method 72
getProperty(), usage example 84
getSelectedIndex, usage example 68
getSelectedItem, usage example 68
getting keyboard input, ASK 44
global variables 60
GLOBALV 47
GLOBALV, CENV 34
glossary 169
GNAME 23
GUIMON
client overview 112
client-server communication 113
functional overview 110
GETDATA Request 115
INDICATE 103
installation instructions 103
installing client 105
installing monitor 103
installing server 106
LISTFILE request 114
LOADBFS 108
monitor overview 110
PERFLOG File Format 117
provided files 108
record format requirements 117
sample program 101
server overview 111

H

handle 59
handle, object 58
handling events, frame 67
Hello World
in Java 38
in NetRexx 38
in REXX 38
host commands, executing 45

I

implement, usage example 69
implementing, methods 69
import, cms.util 94
indexed strings 50
indexed variables 152
inheritable method 58
inheritable properties, usage example 63
inheritable property 60
inheritance 63
input parameters 52, 53
inputStream class 83
InputStreamReader 83
InputStreamReader, 8859_1 87
insensitive case 39, 42
installing
BFSLIST 27
GUIMON client 105
GUIMON server 106
instructions, GUIMON 103
problems 145
the GUIMON monitor 103
instance 59
instance, class 57
instance, object 58
instruction
ADDRESS 45
catch 42
class 62
do 42, 43
exit or return 49
expose 56
finally 42
loop 43
loop over 51
parse 44
procedure expose 56
REXX compared with NetRexx 42
select 43
internationalization, Java 83
interpreter, Java 39
interpreter, REXX 37
introduction, NetRexx syntax 39
isolated, variables 48
ItemListener, usage example 68
itemStateChanged, usage example 69
iterate 43
IWDIR 23

J

Java
-classpath option 33
class case 42
codepage, VM default 84
Compile 31
compiler 38
Hello World sample 38
internationalization 83
interpreter 39
program compiling 26
running a program 31
toolkit 39
vectors 52
Java virtual machine, stopping 49
JC EXEC 31

JIT 39
JNI 94
JNRCMS PACKAGE 100
JNRCMS, installation instructions 100
Just In Time compiler 39

K

keyboard input, translating 50
kill, sequences 77
killing Java virtual machine 144

L

label option 43
layout of the frame 65
LE/370, runtime library 29, 31
line, continuation 40
LINEDEL setting 77
LINK, creating 26
LINK, querying 26
links 25
  BFS 14
  external 14, 25
  symbolic 14, 25
list box
  adding an element 66
  double click event 68
  select event 69
  selected line 68
listen to events 69
listing
  BFS directory tree 27
  directory contents 22, 23, 24
  SFS directories 13
LOADBFS 100
LOADLIB, SCEERUN 29, 31
local variable 60
loop
  instruction 43
  label option 43
  over an array 51
  over instruction 51
looping 43
ls, POSIX Shell command 22, 23, 24

M

main
  input parameters 52, 53
  method 52, 53
  method example 71
  signature 72
method
  arguments 59, 60, 158
  constructor 58, 62
  defined 58
  exit 49
  getProperty 72
method *(continued)*
  implementing 69
  inheritable 58
  introduced 40, 57
  main 52, 53
  nested call 46
  nested calls 47
  NetRexx 155
  parameter checking 49
  private 58
  public 58
  return 49
  signatures 59
  signatures example 66
  static 58
  using to access variables 48
mixed, case environment variable 34
MONREAD
  EXEC 104
  PACKAGE 103
  PROFILE 104
  sample directory entry 104
MOUNT, external 26
mounting BFS file space 14
ms, Java option 143
multi dimension variable 50
mx, Java option 143

N

native data type 41
nested
  function call 46
  method call 46
  NetRexx methods 47
NetRexx
  arithmetic 150
  arrays 153
  class 155
  comments 40
  compared with REXX 37
  compile 29
  compile under XEDIT 30
  compiler 38
  compiler options 30
  compilers 39
  compiling 148
  control statements 149
  data type 41
  declaring variables 41
  exception and error handling 154
  expressions and variables 148
  function calls 46
  GUI 39
  Hello World sample 38
  IF THEN ELSE 149
  language quick start 147
  line continuation 149
  LOOP 150

NetRexx *(continued)*
method calls 151
methods 155
multitasking 39
operators 148
parsing strings 151
positioned 38
program compiling 26
program execute 26
program general structure 61
properties 155
reading console input 149
running a program 31
running under XEDIT 32
runtime library 38, 39
stems 50
strings 151
syntax, basic differences from REXX 40
syntax, compared with REXX 39
syntax, introduction 39
the language 39
things that aren't strings 155
tracing 154
URL for information 160
using objects 155
NetRexxC 148
networking, TCPIP 87
not initialized variable 141
not initialized variables 41
NRC EXEC 26, 29
NRC XEDIT 30
NRC XEDIT, compile time options 62
NRR EXEC 26, 31
NRR XEDIT 32
null variable 42
NullPointerException 141

O

object
attribute 57
handle 58
instance 58
listening to events 69
properties 60
Object Oriented REXX 53
obtaining
host 72
operating system 72
variables, CMS pipelines 56
OO-REXX 53
OPENVM
GETBFS 26
LISTFILE 22, 23
MOUNT 22
PATHNAME 25
SET DIR 22
operating system, obtaining 72
option
binary 150, 159
label 43
strictassign 158
outputStream class 83
OutputStreamReader, 8859_1 87
OutputStreamWriter 83
OVMJAVA package, obtaining 19

P

parent directory 26
parse
ARG 44
arg, REXX statement 52
instruction 44
into words 152
literal patterns 152
positional patterns 152
pull, REXX instruction 44
strings 151
path 25
absolute 8, 9
defined 8
environment variable 33
relative 8, 9
pathname 8, 25
PATHNAME, OPENVM 25
PERFLOG DESCRIBE 104, 118
PIPE BFSSTATE 26
PIPE, BFSSTATE 26
pipelines
calling from Java 96
CMS 37, 56
usage example 115
positioning NetRexx 38
positioning REXX 37
POSIX
.profile 23, 32
shell and utilities 23
POSIX Shell, calling BFSLIST 32
POSIX Shell, calling XEDIT 32
POSIX Shell, system profile 35
POSIXGLIST, CP directory 11
POSIXINFO 23
POSIXINFO, CP directory 10
POSIXOPT, CP directory 10
printLine, ASCII 87
printLine, EBCDIC 87
private
class 58
method 58
variables 48
procedure expose 56
PROFILE EXEC 23
PROFILE XEDIT 21
program interface, AboutFrame 56
programs, canceling 77

properties 62
  definition 60
  examples 57
  explained 63
  inheritable 60
  introduced 57
  NetRexx 155
  static 60, 63
  usage example 63
  when to use 63
public
  class 58
  method 58
  variables 48
push button, select event 69

Q

querying, a LINK 26
questions frequently asked 141

R

reader class 83
reading 75
  BFS 75
  CMS 75
  console 77
readLine, ASCII 87
readLine, EBCDIC 87
readUTF, ASCII 87
relative path 8, 9
renaming BFS directory 26
renaming BFS file 26
return, from method 49
return, instruction 49
reusable class, AboutFrame 55
REXX
  comments 40
  compared with NetRexx 37
  compilers 38
  data type 41
  function calls 45
  Hello World sample 38
  parse arg 52
  positioned 37
  signal on novalue 41
  stems 50
  string type class 41
  the language 37
running, Java program 31
running, NetRexx program 31
runtime
  exec 93
  LE/370 library 29, 31
  NetRexx library 38
  problems 145

S

sample
  directory entry MONREAD 104
  obtaining programs 19
  program GUIMON 101
SCEERUN LOADLIB 29, 31
scope of variables 60
scope, variables 48
scripting language 37, 45
scroll bar, usage example 112
select instruction 43
select label option 43
sensitive, case 22, 42
sequences, kill 77
SET INPUT 50
SET OUTPUT 50
SETCENV EXEC 34
setForeground, usage example 69
setting
  C environment variables 33
  classpath 34
  environment variables 33
  environment variables from CMS 34
SFS directories, listing 13
SFS, file space 13
SG245148 package, obtaining 19
Shell and Utilities, POSIX 23
signal on novalue, REXX 41
signature
  example 66
  for main 72
  method 59
socket, ASCII 87
sorting BFS objects 26
square brackets, entering 50
square brackets, entering from keyboard 50
startup parameters 52, 53
static
  method 58
  property 60, 63
stems 50
  REXX 50
  variable, nullpointerexception 141
stopping Java virtual machine 49
storage required to compile 29
storage requirements 143
stream
  byte 83
  character 83
  definition 83
strictassign, option 158
strings 40
  comparing 42
  compound 50
  functions 46
  indexed 50
  manipulation 46
  parsing 151

subclass, defined 157
subroutines 40, 47
subroutines, external 47
super
  calling class 66
  class 66
  usage example 64, 65
  when to use 65
superclass, defined 157
Symbol, REXX function 42
symbolic links 14, 25
SYSPROF EXEC 35
system
  class 72
  getProperty 72
  profile, CMS 35
  profile, POSIX Shell 35

T

tail 50
TCPIP, networking 87
TERMINAL, LINEDEL setting 77
this variable 59
this, usage example 64, 65
Threads Class not found 142
timezone 33
toolkit, Java 39
tools 21
tracing NetRexx 154
translating
  EBCDIC 94
  keyboard input 50
  strings to lowercase 44
  strings to uppercase 44
tree, directory 26
types, binary 158

U

UID 23
unicode 83
upper method, usage example 44
uppercasing, strings 44
URL, for NetRexx information 160
user defined functions 47
using objects, NetRexx 155
using variables from another EXEC 48
using, array 53

V

variable
  compound 50
  empty variables 41
  global 60
  local 60
  multi dimension 50
  not initialized 141
variable *(continued)*
  null 42
  properties 60
  scope 60
  this 59
variables
  access through methods 48
  array 50
  class 60
  compound 152
  declaring in NetRexx 41
  exposing 56, 60
  indexed 152
  isolated 48
  not initialized variables 41
  obtaining with CMS pipelines 56
  private 48
  public 48
  scope 48
  using from another EXEC 48
Vector, Class 52
vectors, Java 52
virtual storage requirements 29, 143

W

window, closing 67
WindowAdapter, usage example 67
windowClosing, usage example 67
working directory, defining 22
working directory, listing 22
writer class 83
writeUTF, ASCII 87
writing 78
writing, CMS 78

X

XEDIT
  a BFS file 26
  an ASCII BFS file 26
  calling from POSIX Shell 32
  compile NetRexx program 30
  PROFILE 21
  running NetRexx program 32

ITSO Redbook Evaluation

VM/ESA Network Computing with Java and NetRexx
SG24-5148-00

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

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
__Customer __Business Partner __Solution Developer __IBM employee
__None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____________

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:
____

____

____

____

What other redbooks would you like to see published?
____

____

____

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

____

____

____

____

© Copyright IBM Corp. 1998

SG24-5148-00
Printed in the U.S.A.

Artwork Definitions

id	File	Page	References	
WOLOGO	5148SU	i		
WOLOGOS	5148SU	i		
TILOGO	5148SU	i		
TILOGOS	5148SU	i		

Table Definitions

id	File	Page	References	
R2	REDB$EVA	193		193
R1	REDB$EVA	193		193, 193

Figures

id	File	Page	References	
SFSFIG	5148CH02	4	2	3
OVIEPSX	5148CH02	10	7	14
FIG6	5148CH02	12	8	12
ABFR1	5148CH05	55	9	55, 57, 138
ABFRFG2	5148CH05	61	10	60, 61
ABFRFG3	5148CH05	63	11	63
ABFRFG4	5148CH05	64	12	64, 67, 68, 73
ABFRFG5	5148CH05	66	13	65, 66
ABFRFG6	5148CH05	67	14	65, 67
ABFRFG7	5148CH05	68	15	68
ABFRXCL	5148CH05	70	16	70
ABFRMAI	5148CH05	72	17	71
ABFRWIN	5148CH05	73	18	73
READ	5148CH06	75	19	75
WRITE	5148CH06	78	20	78
FACT	5148CH06	79	21	79
ASCIFG1	5148CH07	81	22	81
ASCIFG2	5148CH07	82	23	

81

TCPCL	5148CH08	88	24	88
TCPSERV	5148CH08	89	25	88
SRVHAND	5148CH08	91	26	91
GUIMFG1	5148CH10	101	27	101
GUIMFG2	5148CH10	102	28	102
GUIMFG3	5148CH10	102	29	102
GUIMLST	5148CH10	115	30	114
GUIMGTD	5148CH10	116	31	115
GUIMIND	5148CH10	117	32	116
GUIMXYZ	5148CH10	119	33	118
NETW1	5148CH11	125	34	125
NETW11	5148CH11	129	35	128, 129, 129
NETW12	5148CH11	129	36	129, 129
NETW13	5148CH11	130	37	129
NETW2	5148CH11	131	38	131
NETW3	5148CH11	134	39	133
NETW4	5148CH11	134	40	134
NETW5	5148CH11	135	41	134
NETW6	5148CH11	136	42	135
NETW7	5148CH11	137	43	137
NETW8	5148CH11	138	44	137
NETW9	5148CH11	138	45	138

Headings

id	File	Page	References
REDBCOM	REDB$COM	xiv	Comments Welcome
INTR	5148CH01	1	Chapter 1, Introduction
OVIE	5148CH02	3	Chapter 2, Overview of NetRexx and Java on VM/ESA
OVERSFS	5148CH02	3	2.2, Overview of the SFS 16
OVIEINS	5148CH02	16	2.7, Installing Java and NetRexx without the Shell and Utilities 33, 145
OVMJAVA	5148CH02	19	2.7.1.4, Download and Install the OVMJAVA Package
SAMPGMS	5148CH02	19	2.7.1.5, Download and Install the SG245148 Package 2, 19, 21, 94, 100, 103, 107
TOOL	5148CH03	21	Chapter 3, Tools Used During the Project
BFSLIST	5148CH03	22	3.2, BFSLIST - Listing the Contents of a BFS Directory
TOOLMNT	5148CH03	23	Automatic Mount 106
BFSTREE	5148CH03	27	3.3, BFSTREE - Listing a BFS Directory Tree
TOOLNRC	5148CH03	29	3.4, NetRexx Compile 19
TOOLNRX	5148CH03	30	3.4.2, NRC XEDIT - NetRexx Compile 62
TOOLJC	5148CH03	31	3.5, JC EXEC - Java Compile 19
TOOLNRR	5148CH03	31	3.6, NetRexx Run 19
TOOLENV	5148CH03	33	3.8, SETCENV - Setting C Environment Variables 19, 107
TOOLCLP	5148CH03	33	3.8.1, Important Environment Variables 29, 31, 142
TOOLMCP	5148CH03	34	3.8.3, More About Classpath 33
REXX1	5148CH04	37	Chapter 4, Comparing REXX to NetRexx
REXXADR	5148CH04	45	The ADDRESS instruction 45
REXXSUB	5148CH04	47	4.3.6, Subroutines and User Defined Functions 56
REXXSTM	5148CH04	50	4.3.8, Stems - Array Variables - Indexed Strings 141
ABFR1	5148CH05	55	Chapter 5, AboutFrame, a Reusable Class 40, 48, 113
ABFRABF	5148CH05	60	5.6, AboutFrame: the Class Definition 57
ABFROVR	5148CH05	61	5.6.1, AboutFrame: Overview of the Program
ABFR2WN	5148CH05	73	5.6.6, Avoiding Empty Frames 65
FRWRCMS	5148CH06	75	6.1.1, Reading CMS Character Data Files 99
FRWCTLS	5148CH06	77	6.2.1, Useful Control Sequences 145
FRWWCMS	5148CH06	78	6.3.1, Writing CMS Character Data Files 99
ASCI1	5148CH07	81	Chapter 7, Code Pages - ASCII < > EBCDIC Issues

ID	File	Page	Heading	References
				129
JAVACP	5148CH07	84	7.4, VM Java Codepage	87
FRWSRVR	5148CH08	88	8.3, Simple TCP/IP Server	111
JCMS	5148CH09	93	Chapter 9, Java and CMS	45
JCMSRUN	5148CH09	93	9.1.1, Using Runtime.exec()	96
JCMSPIP	5148CH09	96	9.4, Running CMS Pipelines with NetRexx	106, 107, 111
JCMSINS	5148CH09	100	9.5, Installation Instructions	106, 107
GUIM1	5148CH10	101	Chapter 10, The GUIMON Sample Program	65, 130
GUIMCMS	5148CH10	107	10.2.4, Installing Client and Server Files	106
GUIMCS	5148CH10	113	10.4, GUIMON - the Client-Server Communication	99, 111, 111, 112, 113
GUIMREC	5148CH10	117	10.5, GUIMON Record Format Requirements	110
GUIMPFL	5148CH10	117	10.5.1.1, The PERFLOG File Format	
GUIMPFD	5148CH10	118	10.5.1.2, The PERFLOG DESCRIBE File	114
CHP3	5148CH11	121	Chapter 11, Running NetRexx and Java Applications on a Network Station	
NETWS	5148CH11	122	11.3, Support Delivery Mechanism	
NETWJNR	5148CH11	128	11.8, Setting up to Run Java and NetRexx Programs	105
NETWCPS	5148CH11	128	11.8.1.1, Copying the NetRexx Classes with the Shell	
NETWAE	5148CH11	129	11.8.1.2, Codepages - ASCII <> EBCDIC for Network Stations	85
NETWCOP	5148CH11	129	11.8.1.3, Copying the NetRexx and Cp1047 Classes without the Shell	
NETWLFS	5148CH11	131	11.10, How to Tailor the Local File System	130, 135, 136
NETWPRF	5148CH11	133	11.10.1, Performance Considerations	128, 136
NETWBAR	5148CH11	133	11.11, Using NSM to Add a Java Application to the Menu Bar	130, 136
FAQ	5148AX01	141	Appendix A, Frequently Asked Questions	
FAQEM	5148AX01	142	A.4, Error Messages Not Always Very Accurate	142
FAQSTG	5148AX01	143	A.9, Virtual Storage Requirements	29
FAQRUN	5148AX01	145	A.11, Runtime Problems	
FAQINS	5148AX01	145	A.12, Installation Problems	
NROV1	5148AX02	147	Appendix B, NetRexx Language Quick Start	
NOTICES	SG245148 SCRIPT	161	Appendix C, Special Notices	ii
BIBL	5148BIBL	163	Appendix D, Related Publications	
REDBCDR	REDB$BIB	163	D.2, Redbooks on CD-ROMs	
ORDER	REDB$ORD	165	How to Get ITSO Redbooks	

id	File	Page	References
			163
REDBIBM	REDB$ORD	165	How IBM Employees Can Get ITSO Redbooks
REDBCUS	REDB$ORD	166	How Customers Can Get ITSO Redbooks
REDBFOR	REDB$ORD	167	IBM Redbook Order Form
REDBEVA	REDB$EVA	187	ITSO Redbook Evaluation xiv

Index Entries

id	File	Page	References
ABFIND	5148VARS	i	(1) AboutFrame 55, 55, 56, 60, 61, 63, 63, 64, 66, 67, 68, 71, 71
ARRIND	5148VARS	i	(1) array 50, 51, 51, 51, 53, 53, 63, 63, 65, 66, 153
ASCIND	5148VARS	i	(1) ASCII 26, 26, 81, 87, 87, 87, 87, 94
BFSDIND	5148VARS	i	(1) BFS directory 26, 26, 26, 27, 31
BFSFIND	5148VARS	i	(1) BFS file 24, 26, 26, 26, 26, 26, 26
BFSIND	5148VARS	i	(1) BFS 13, 14, 14, 22, 23, 24, 26, 26, 79, 79
BFSLIND	5148VARS	i	(1) BFSLIST 27, 27, 27, 27, 32
BININD	5148VARS	i	(1) binary 79, 79, 150, 158, 159, 159
CASIND	5148VARS	i	(1) case 22, 34, 34, 38, 39, 42, 42
CLIND	5148VARS	i	(1) class 38, 39, 40, 41, 42, 52, 57, 57, 58, 58, 60, 60, 62, 63, 66, 72, 91, 155, 156
CMSIND	5148VARS	i	(1) CMS 35, 37, 45, 56, 78, 93, 94, 94, 95, 96
CMSPIND	5148VARS	i	(1) CMSPipe 96, 97, 98, 115
CODIND	5148VARS	i	(1) codepage 82, 84, 84, 85
COMIND	5148VARS	i	(1) commands 24, 26, 32, 33
COMPIND	5148VARS	i	(1) compile 26, 26, 29, 29, 30, 30, 31, 38, 38, 38, 39, 39
COMVIND	5148VARS	i	(1) compound variable 118, 141
CONSIND	5148VARS	i	(1) console 77, 77, 77, 77, 77
CONTIND	5148VARS	i	(1) constructor 58, 62, 64, 157
COPRIND	5148VARS	i	(1) comparing 37, 42, 42
CREAIND	5148VARS	i	(1) creating 11, 11, 11, 14, 14, 14, 14, 26
DATIND	5148VARS	i	(1) data type 41, 41, 41, 41, 142, 158, 159
DIRIND	5148VARS	i	(1) directory 13, 22, 23, 23, 24, 26, 26, 31
EBCDIND	5148VARS		

i (1) EBCDIC
81, 87, 87, 87, 94

ENRIND 5148VARS
i (1) enrolling
11, 11, 13, 14

ENVIND 5148VARS
i (1) environment variables
33, 33, 33, 34

EXECIND 5148VARS
i (1) executing
26, 45, 45

EXTIND 5148VARS
i (1) external
14, 25, 26, 47

FILIND 5148VARS
i (1) file space
13, 14, 15, 15

FITIND 5148VARS
i (1) fitting
96, 96, 96, 97, 97, 97, 98, 98

FRAMIND 5148VARS
i (1) frame
65, 65, 65, 67, 67, 73

FUNCIND 5148VARS
i (1) function calls
45, 46, 46

GUIMIND 5148VARS
i (1) GUIMON
101, 103, 103, 103, 105, 106, 108, 108, 110, 110, 111,
112, 113, 114, 115, 117, 117

HELLIND 5148VARS
i (1) Hello World
38, 38, 38

INSTIND 5148VARS
i (1) installing
27, 103, 103, 105, 106, 145

INSIND 5148VARS
i (1) instruction
42, 42, 42, 42, 43, 43, 43, 44, 45, 49, 51, 56, 56, 62

JAVAIND 5148VARS
i (1) Java
26, 31, 31, 33, 38, 38, 39, 39, 42, 52, 83, 84

LINKIND 5148VARS
i (1) links
14, 14, 14, 25, 25

LISTIND 5148VARS
i (1) list box
66, 68, 68, 69

LSTIND 5148VARS
i (1) listing
13, 22, 23, 24, 27

LOOPIND 5148VARS
i (1) loop
43, 43, 51, 51

MAININD 5148VARS
i (1) main
52, 52, 53, 53, 71, 72

METHIND 5148VARS
i (1) method
40, 46, 47, 48, 49, 49, 49, 52, 53, 57, 58, 58, 58, 58, 58,
58, 59, 59, 60, 62, 66, 69, 72, 155, 158

MONIND 5148VARS
i (1) MONREAD
103, 104, 104, 104

NESTIND 5148VARS
i (1) nested
46, 46, 47

NETIND 5148VARS
i (1) NetRexx
26, 26, 29, 30, 30, 31, 32, 37, 38, 38, 38, 38, 39, 39, 39,
39, 39, 39, 39, 40, 40, 41, 41, 46, 50, 61, 147, 148, 148,
148, 149, 149, 149, 149, 150, 150, 151, 151, 151, 153,
154, 154, 155, 155, 155, 155, 155, 160

OBJIND 5148VARS
i (1) object
57, 58, 58, 60, 69

OBTIND 5148VARS
i (1) obtaining
56, 72, 72

OPENIND 5148VARS
i (1) OPENVM
22, 22, 22, 23, 25, 26

OPTIND 5148VARS
i (1) option
43, 150, 158, 159

PARSIND 5148VARS
i (1) parse
44, 44, 44, 52, 151, 152, 152, 152

PATHIND 5148VARS
i (1) path
8, 8, 8, 9, 9, 33

PIPIND 5148VARS
i (1) pipelines
37, 56, 96, 115

POSIND 5148VARS
i (1) POSIX
23, 23, 23, 32

PROPIND 5148VARS
i (1) properties
57, 57, 60, 60, 60, 63, 63, 63, 63, 155

PRIVIND 5148VARS
i (1) private
48, 58, 58

PUBIND 5148VARS
i (1) public
48, 58, 58

READIND 5148VARS
i (1) reading
75, 75, 77

REXXIND 5148VARS
i (1) REXX
37, 37, 37, 38, 38, 40, 41, 41, 41, 45, 50, 52

RUNIND 5148VARS
i (1) runtime
29, 31, 38, 93, 145

SAMPIND 5148VARS
i (1) sample
19, 101, 104

SETIND 5148VARS
i (1) setting
33, 33, 34, 34

SIGIND 5148VARS
i (1) signature
59, 66, 72

STATIND 5148VARS
i (1) static
58, 60, 63

STEMIND 5148VARS
i (1) stems
50, 141

STRIND 5148VARS
i (1) strings
42, 46, 46, 50, 50, 151

STRMIND 5148VARS
i (1) stream
83, 83, 83

SUPIND 5148VARS
i (1) super
64, 65, 65, 66, 66

SYSIND 5148VARS
i (1) system
35, 35, 72, 72

TRANIND 5148VARS
i (1) translating
44, 44, 50, 94

VARIND 5148VARS
i (1) variable
41, 42, 50, 50, 59, 60, 60, 60, 60, 141

VARSIND 5148VARS
i (1) variables
41, 41, 48, 48, 48, 48, 48, 48, 48, 50, 56, 56, 60, 60, 152,
152

XEDIND 5148VARS
i (1) XEDIT
21, 26, 26, 30, 32, 32

Footnotes

id	File	Page	References
BFSCNV	5148CH02		
		14	1
			14
ABFRPRO	5148CH05		
		63	2
			63

Tables

id	File	Page	References
CMDTAB	5148CH02		
		11	1
			11

Processing Options

Runtime values:

- Document fileid .... SG245148 SCRIPT
- Document type .... USERDOC
- Document style .... REDBOOK
- Profile EDFPRF40
- Service Level 0022
- SCRIPT/VS Release 4.0.0
- Date 98.11.19
- Time 04:55:40
- Device 3820A
- Number of Passes 4
- Index .... YES
- SYSVAR D .... YES
- SYSVAR G .... INLINE
- SYSVAR X .... YES

Formatting values used:

- Annotation .... NO
- Cross reference listing .... YES
- Cross reference head prefix only .... NO
- Dialog .... LABEL
- Duplex .... YES
- DVCF conditions file .... (none)
- DVCF value 1 .... (none)
- DVCF value 2 .... (none)
- DVCF value 3 .... (none)
- DVCF value 4 .... (none)
- DVCF value 5 .... (none)
- DVCF value 6 .... (none)
- DVCF value 7 .... (none)
- DVCF value 8 .... (none)
- DVCF value 9 .... (none)
- Explode .... NO
- Figure list on new page .... YES
- Figure/table number separation .... YES
- Folio-by-chapter .... NO
- Head 0 body text .... Part
- Head 1 body text .... Chapter
- Head 1 appendix text .... Appendix
- Hyphenation .... NO
- Justification .... NO
- Language .... ENGL
- Keyboard 395
- Layout .... OFF
- Leader dots .... YES
- Master index .... (none)
- Partial TOC (maximum level) 4
- Partial TOC (new page after) .... INLINE
- Print example id′s .... NO
- Print cross reference page numbers .... YES
- Process value .... (none)
- Punctuation move characters ....,
- Read cross-reference file .... (none)
- Running heading/footing rule .... NONE
- Show index entries .... NO
- Table of Contents (maximum level) 3

Table list on new page .... YES
Title page (draft) alignment .... RIGHT
Write cross-reference file .... (none)

Imbed Trace

Page 0	5148SU
Page 0	5148VARS
Page 0	REDB$POK
Page i	REDB$ED1
Page i	5148EDNO
Page i	REDB$ED2
Page xiii	5148ABST
Page xiii	5148ACKS
Page xiv	REDB$COM
Page xiv	5148IMBD
Page xiv	5148CH01
Page 2	5148CH02
Page 20	5148CH03
Page 35	5148CH04
Page 53	5148CH05
Page 73	5148CH06
Page 79	5148CH07
Page 85	5148CH08
Page 92	5148CH09
Page 100	5148CH10
Page 120	5148CH11
Page 140	5148AX01
Page 146	5148AX02
Page 161	5148SPEC
Page 161	REDB$SPE
Page 162	5148TMKS
Page 162	5148BIBL
Page 163	REDB$BIB
Page 164	REDB$ORD
Page 167	5148GLOS
Page 181	5148ABRV
Page 192	REDB$EVA