

DB2® Server for VM



System Administration

Version 6 Release 1

DB2® Server for VM



System Administration

Version 6 Release 1

Note!

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

This book is also provided as an online book that can be viewed with the IBM® BookManager® READ and IBM Library Reader™ licensed programs.

First Edition (December 1998)

This edition, SC09-2657, applies to Version 6 Release 1, of the IBM DATABASE 2™ Server for VSE & VM Program 5648-A70, and to all subsequent releases of this product until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to: IBM Canada Ltd. Laboratory, Information Development 2G/345/1150/TOR, 1150 Eglinton Ave East, North York, Ontario, Canada. M3C 1H7

You can also send your comments by facsimile to (416) 448-6161 addressed to the attention of the RCF Coordinator. If you have access to Internet, you can send your comments electronically to torrcf@ca.ibm.com; IBMLink™, to [toribm\(torrcf\)](mailto:toribm(torrcf)@ibm.com); IBM/PROFS®, to [torolab4\(torrcf\)](mailto:torolab4(torrcf)@ibm.com); IBMMAIL, to [ibmmail\(caibmwt9\)](mailto:ibmmail(caibmwt9)@ibm.com); or through our home page at <http://www.software.ibm.com/data/db2/vse-vm>

If you choose to respond through Internet, please include either your entire Internet network address, or a postal address.

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

© Copyright International Business Machines Corporation 1987, 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.

Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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, U.S.A.

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 Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario Canada M3C 1H7. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include 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.

Programming Interface Information

This manual is intended to help system administrators plan and maintain the database manager.

This manual also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by the database manager.

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

General-use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

General-Use Programming Interface

General-use Programming Interface and Associated Guidance Information...

End of General-Use Programming Interface

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of the

database manager. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

```
_____ Product-Sensitive Programming Interface _____
Product-sensitive Programming Interface and Associated Guidance Information...
_____ End of Product-Sensitive Programming Interface _____
```

Trademarks

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

AIX/6000	BookManager
CICS	CICS/VSE
DATABASE 2	DB2
DB2 for AIX	DB2 Server for VM
DB2 Server for VSE	DB2 Server for VSE & VM
Distributed Relational Database Architecture	DRDA
IBM	Library Reader
OS/2	OS/400
SQL/DS	System/370
System/390	
Virtual Machine/Enterprise Systems Architecture	VM/ESA
VSE/ESA	VTAM

Lotus and Lotus Notes are trademarks of Lotus Development Corporation in the United States and/or other countries.

Other company, product, and service names may be trademarks or service marks of others.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

Contents

Notices	iii
Programming Interface Information	iii
Trademarks	iv

About This Manual	xi
Organization of This Manual	xi
Syntax Notation Conventions	xiii
SQL Reserved Words	xvii

Summary of Changes for DB2 Version 6

Release 1	xix
Enhancements, New Functions, and New Capabilities	xix
DRDA® RUOW Application Requestor for VSE (Online)	xix
Stored Procedures	xix
TCP/IP Support for DB2 Server for VM	xx
New Code Page and Euro Symbol Code Page Support	xx
DataPropagator™ Capture	xxi
QMF for VM, QMF for VSE, and QMF for Windows®	xxi
RDS Above the Line	xxi
Combining of NLS Feature Installation Tapes with Base Product Installation Tape	xxi
Control Center Feature	xxii
Data Restore Feature	xxii
DB2 REXX SQL Feature	xxii
Reliability, Availability, and Serviceability Improvements	xxii
Migration Considerations	xxii
Library Enhancements	xxiii

Chapter 1. Planning for Installation	1
Operating System Overview	1
Virtual Machine Overview	1
Components of the Relational Database Management System	2
Software Requirements	4
Virtual Storage Requirements	5
Database Machine Size	5
Service Machine Size	5
User Machine Size	6
Hardware Requirements	6
Real Storage Requirements	6
DASD Space Requirements	7
Tape Requirements	10
Display Terminal Requirements	10
Considerations When Defining a Database Machine and Generating a Database	11

Considerations When Adding Directory Control Statements	11
Considerations When Loading IBM-Supplied Files	11
Considerations When Generating a Database	11
Considerations When Defining a Service Machine	11
Updating the Service Machine VM Directory	11
Considerations When Loading IBM-supplied Files	11
Defining User Machines	12
Defining Saved Segments	12
Setting Up the CMS Communications Directory	12
Updating the SNA NETID File	13

Chapter 2. Planning for Database Generation

Generation	15
Database Generation Parameters	15
Defining Database Directory Size	16
Defining the Database Log	18
Establishing Database Capacity Parameters	20
Establishing Initial Dbspace Requirements	21
Determining Initial Dbextent Requirements	24
Choosing an Application Server Name and VM Resource Identifier	25
Choosing the Application Server Default CHARNAME and CCSID	26
Choosing the Application Server Default Character Subtype	28
Choosing the Default CHARNAME and CCSID for Application Requesters	29
Preparing for Database Regeneration	29
Database Generation Worksheet	30

Chapter 3. Planning for Database Migration

Migration	33
Migration Considerations	34
Increasing the HELPTTEXT Dbspace	34
Migrating from Version 3 Release 1	34
Considerations for Invalid Indexes	34
Conversion of Packages	35
Migrating from Version 3 Release 2	35
Choosing an Application Server Default CHARNAME	35
Choosing the Default CHARNAME for All Application Requesters	38
Considerations for Mixed Primary Keys with Field Procedures	39

Migrating from Version 3 Release 3	39
Considerations for EXPLAIN Tables	39
Considerations for VSE Guest Sharing	39
Considerations for the VM Data Spaces Support (VMDSS)	39
Migrating from Version 3 Release 4	39
Considerations for Assembler Even Precision Packed Decimal	39
Considerations for SQLSTATE Changes for SQL92 Support	40
Migrating from Version 3 Release 5	40
Considerations for Uncommitted Read	40
Considerations for VMSES/E	40
Considerations for Support of ESA-mode Processors Only	40
Considerations for the Renaming of the Product	40
Considerations for the Removal of the User Facility Subset	41
Migrating from Version 5 Release 1	41
Considerations for RDS Above 16M	41
Considerations for TCP/IP	41
Release Coexistence Considerations	41
Version Compatibility Matrix	41
Migrating from a VSE to a VM Operating System	42
Moving a Database from a VSE to a VM Operating System	43
Choosing a VM Resource Identifier	43
Converting Data in the Database	43
Converting Packages	43
Converting Programs	44
VSE Databases Coexisting under VM	44
Migrating from a VM/XA to a VM/ESA Environment	44
Delaying the Directory and Database Name Changes	44
Setting up the Database Machine Directory Entry	45
Example of a Database Machine Directory with Multiple Databases	46
Setting Up the User Machine Directory Entry	47
Database Naming Considerations	49
Migrating from a VM/SP to a VM/ESA Operating System	49
Installing Another IBM VM System on Your Processor	49
Moving a Database	49
Moving a VM Application Server from One User ID to Another	50
Converting a Service Machine to a Database Machine	53

Changing the Server Name and Resource Identifier	53
--	----

Chapter 4. Planning for Operation of the Database Manager	57
Starting the Application Server	57
The Database Operator	57
Multiple User Mode Initialization Parameters	58
Single User Mode Initialization Parameters	77
Tape Support	79
General File Support	82
Starting the Application Server in Multiple User Mode	83
Starting the Application Server in Single User Mode	86
Overriding Initialization Parameters	93
Creating a Parameter File	94
Running the Database Manager	95
Operating Modes	95
Disconnecting the Database Machine	96
Stopping the Application Server	97
Taking an Archive	97
Verifying the Directory	99
Online Support Considerations for VSE Guest Sharing	99
A Note about Minidisk Passwords	100
Inter-Machine Communications	100

Chapter 5. Operating the Online Support for VSE Guest Sharing	103
Operating VSE Guest Sharing	103
Operator Responsibilities	104
Starting the Application Server	105
Starting the Online Resource Adapter -- The CIRB Transaction	106
Adding Connections -- The CIRA Transaction	113
Automatic Restart Resynchronization	116
Changing the Default Server -- The CIRC Transaction	122
Removing Connections -- The CIRR Transaction	123
Displaying Information -- The CIRD Transaction	126
Stopping the Online Support -- The CIRT Transaction	135
Password Implications on Online Resource Adapter Termination	140

Chapter 6. Maintaining Database Security	143
Communications and System Security	144
Session-Level Security	144
Conversation-Level Security	144
VM Directory Control Statements	146

User ID Translation	147	Archiving Procedures	211
Minidisk Protection	148	Performing Database Archives With	
CMS Restrictions	148	Database Manager Facilities	211
System and DB2 Server for VM Operator		Example of an SQLEND ARCHIVE	213
Console Considerations	149	Performing Database Archives With User	
Access Control to ISQL on a VSE Guest	149	Facilities	215
Chapter 7. Managing Database Storage	151	Performing Log Archives	216
Storage Concepts	151	Example of an SQLEND LARCHIVE	218
How Information is Stored in Dbspaces	152	Labeling Your Archive Tapes	223
Adding Dbspaces to the Database	153	Recovery Procedures	223
Considerations for Adding Dbspaces	156	Restarting Procedures	223
Example of Adding a Dspace to a		Restoring the Database	224
Database	158	Restarting from Failure of a Database	
Expanding the Database Directory	159	Restore	228
Acquiring Dbspaces for Packages	161	Restarting from a System Failure While	
Managing Storage Pools	164	Archiving	230
Design Considerations for Storage Pools	164	Restarting from Failure of a Database	
Monitoring Storage Pools	165	Generation or COLDLOG Operation	230
Maintaining Storage Pools	165	Relocating the Database Manager	231
Running the SQLADBEX EXEC	171	Replacing a Minidisk Using DASD Dump	
Moving Dbextents	177	Restore	231
Moving Log Disks	179	Replacing a Database Minidisk	232
Chapter 8. Saved Segments	181	Replacing a Log Minidisk	235
Using Saved Segments for Components	181	Recovering to a Secondary System	236
Example 1	185	Chapter 10. Special Topics in Recovery	
Example 2	186	Design	237
Example 3	186	Switching Log Modes	237
Example 4	186	From LOGMODE=A	237
Defining Saved Segments	187	From LOGMODE=L	238
.	190	From LOGMODE=Y or N	239
.	195	Using Dual Logging	239
Running in User Free Storage after Using		Using the VM DUPLEX Command	239
Default Saved Segments	200	Reconfiguring and Reformatting the Logs	240
ARISNLSC MACRO	201	Log Reconfiguration	240
Chapter 9. Making Backups and		Log Reformatting	241
Recovering from Failures	203	Running the SQLLOG EXEC	241
Understanding Recovery Concepts	203	Switching Log Data between Logs	244
What is a Logical Unit of Work?	204	History Area	244
What is a Log?	204	Nonrecoverable Storage Pools	250
What is a Checkpoint?	205	Characteristics of Dbspaces in	
What Happens after a System Failure?	205	Nonrecoverable Storage Pools	251
What is an Archive?	206	Data That Can be Placed in	
Recovering from DASD Failures that		Nonrecoverable Storage Pools	254
Damage the Database	207	Data That Should Not Be Placed in	
Recovering from DASD Failures that		Nonrecoverable Dbspaces	257
Damage a Log	208	Setting Up Nonrecoverable Storage Pools	
Recovering from DASD Failures that		and Dbspaces	257
Damage the Database and Log	208	Querying for Nonrecoverable Storage	
Establishing DASD Recovery Procedures	208	Pools and Dbspaces	258
Choosing a Log Mode	208	Chapter 11. Using the Accounting Facility	261
Backing Up the History Area	211	Where to Find More about VM Accounting	261
		Preparing to Use the Accounting Facility	261

Starting the Accounting Facility	262	Changing the Subtype Attribute of an Existing Column	340
Generation of Accounting Records	263	Setting the Application Requester Default CHARNAME and CCSIDs	341
Supplying Accounting Data from DRDA Applications	265	Setting the Default CHARNAME and CCSIDs for All Application Requesters	341
Formats of the Accounting Records	265	Setting the Default CHARNAME and CCSIDs for an Application Requester	342
Initialization Records	266	Setting the Application Server Default Character Subtype	342
Operator and Checkpoint Records	267	Setting the DBCS Option for the Application Server	343
Termination Records	267	Setting DBCS Option for Application Requestors	343
CMS User Records	268	Setting the DBCS Option for all Application Requesters	344
Remote User Records	269	Setting the DBCS Option for an Application Requester	344
VSE Guest User Records	269	EUC Conversions	344
Maintaining Accounting Data	270	Examples of Setting Values for an Installation	345
Considerations for an Accounting Dbspace	271	Example 1	345
Tables to Hold Accounting Data	271	Example 2	346
Loading the Accounting Data	274	Identifying Classification and Translation Tables for a CCSID	348
Chapter 12. Planning and Implementing Configurations	277	National Language Support for Messages and HELP Text	348
Configuration Concepts	277	CMS HELP Text Files	351
Reasons for Adding a Database Machine	277	National Language Messages in a VSE Guest Sharing Environment	351
Databases in a TSAF Collection or an SNA Network	280	Defining Message Repositories as Saved Segments	351
Adding Service Machines	282	Chapter 14. Creating Installation Exits	355
Types of Database Machines	284	Supplying Account Numbers for Users	355
Primary Database Machines	286	How the ARIUXIT Module Works	356
Why Add a Database Machine?	286	Coding Your Own Accounting Exit	360
Adding a Primary Database Machine	287	Installing Your Version of ARIUXIT	366
Adding a Secondary Database Machine	292	Service Considerations for ARIUXIT	368
Adding a Service Machine	294	Defining Your Own Datetime Format	368
Defining Additional User Machines	294	Datetime Formats	368
Adding a Database	297	How Datetime Exits Work	369
VSE Guest Sharing Configuration	314	Coding Your Own Datetime Exit	372
Chapter 13. Choosing a National Language and Defining Character Sets	319	Installing Your Version of ARIUXDT or ARIUXTM	378
Considerations when changing default CHARNAME and CCSID	320	Updating the SYSTEM.SYSOPTIONS Catalog Table	379
Changing from pre-Euro CHARNAME to Euro-compatible CHARNAME	321	Coding Your Own TRANSPROC Exit	380
Using Alternative Character Sets	322	380
Hexadecimal Values of the Sample Character Sets	323	Coding Your Own Cancel Exit	383
Specifying an IBM-Supplied Character Set at Run Time	329	Resource Adapter Cancel Support	384
Using Double-Byte Character Set (DBCS) Identifiers Containing DBCS Characters	331	RMXC (Resource Adapter Cancel Exit Control)	384
Constants and Data Containing DBCS Characters	332	Field Procedures	387
CCSID Conversion	333	Specifying the Field Procedure	388
Determining CCSID Values	337		
Setting the Application Server Default CHARNAME and CCSIDs	338		
Changing the CCSID Attribute of an Existing Column	340		

When Field Procedures are Called	388	Storage Capacities of IBM DASD Devices	449
General Considerations for Writing Field Procedures	389	Determining Equivalent Minidisk Sizes on Different Device Types	452
A Warning about Blanks	390	Relationship of Megabytes to 4-Kilobyte Pages	454
Maintaining Field Procedures	390	Estimating Directory Space Requirements	455
Recovering from Abends in Exits	390	Estimating Storage Pool Requirements	455
Security with Field Procedures	390	Estimating SYS0001 Dbspace Requirements	456
Field Procedures for Cultural Sorts	390	SYS0001 Storage Estimating General Formula Assumptions	456
Field Procedure Interface to the Database Manager	392	Derivation of the General Formula for SYS0001 Storage Estimating	460
Field-Definition (Function Code 8)	396	Formula for SYS0001 Storage Estimating	461
Field-Encoding (Function Code 0)	397	Examples of Using the SYS0001 Storage Estimating Formula	461
Field-Decoding (Function Code 4)	399	Modifying the SYS0001 Storage Estimating General Formula	463
Chapter 15. Using a DRDA Environment	411	Estimating ISQL Dbspace Requirements	465
.	411	Estimating Dbspace Sizes for Routines	465
Benefits of Using the DRDA Protocol	411	Estimating Dbspace Size for Stored SQL Statements (Stored Queries)	466
Added Responsibilities in Using the DRDA Protocol	412	Appendix C. Maximum Values	469
Preparing to Implement DRDA	412	Database Manager Maximum Values	469
Installing and Removing the DRDA Code	413	Database Maximum Values	469
Steps to Install or Remove the DRDA Code	414	Appendix D. Updating SYSTEM.SYSSTRINGS	471
Using the DBS Utility and ISQL on Other Application Servers	417	Appendix E. Defining Your Own Character Set	475
Creating the DBS Utility Package	417	Step 1: Identify All Characters in Your Character Set	476
Creating the ISQL Package	418	Step 2: Classify the Characters	477
Types of Access to Distributed Data	419	Step 3: Determine Translation Characters	486
Remote Unit of Work	419	Step 4: Update the SYSTEM.SYSCHARSETS Catalog Table	487
The Distributed Two-Phase Commit Process	419	Step 5: Update the SYSTEM.SYSCCSIDS Catalog Table	488
Two-Phase Commit Processing	420	Step 6: Update the SYSTEM.SYSSTRINGS Catalog Table	489
Operator Commands	424	Step 7: Update the CCSID-Related CMS Files	490
CRR Operator Commands	424	Appendix F. Macro List	491
Resynchronization	425	Appendix G. Service and Maintenance Execs	493
Chapter 16. Using TCP/IP with DB2 Server for VM	435	ARISAVES EXEC	493
Preparing the Application Server to use TCP/IP	435	ARISPDFC EXEC	495
Preparing the Application Requester to use TCP/IP	437	Authorization	495
Security Considerations for TCP/IP	438	Syntax	495
Application Requester	440	Description	495
Appendix A. Virtual and Real Storage Requirements	443	Notes:	496
Initial Virtual Storage Requirements of Components	443		
User Machine Components	445		
Use of SELECT	446		
Use of Routines or EXECs	447		
Appendix B. Estimating Database Storage	449		

SQLBOOTS EXEC	496	Detailed Notes on V2R2-V2R1	
Authorization	496	Incompatibilities	509
Syntax	496	V3R1 and V2R2 Incompatibilities	510
Description	496	Detailed Notes on V3R1-V2R2	
SQLGENLD EXEC	496	Incompatibilities	515
Authorization	497	V3R2 and V3R1 Incompatibilities	521
Syntax	497	Detailed Notes on V3R2-V3R1	
Description	497	Incompatibilities	525
Appendix H. DRDA Considerations	499	V3R3 and V3R2 Incompatibilities (VM Only)	527
Omissions from the Standards	499	Detailed Notes on V3R3-V3R2	
Extensions to the Standards	499	Incompatibilities	533
DB2 Server for VSE & VM Facility		V3R4 and V3R3 Incompatibilities (VM Only)	534
Restrictions	500	Detailed Notes on V3R4-V3R3	
Appendix I. Incompatibilities Between		Incompatibilities	538
Releases	503	V3R5 and V3R4 Incompatibilities	539
Definition of an Incompatibility	503	V5R1 and V3R5 Incompatibilities	540
Impact on Existing Applications	503	V6R1 and V5R1 Incompatibilities	540
V2R1 and V1R3.5 Incompatibilities	504	Bibliography	541
V2R2 and V2R1 Incompatibilities	506	Index	545

About This Manual

This manual describes how to carry out system planning and administration tasks for the DB2 Server for VM database manager that is:

- On a VM/ESA™ operating system (Virtual Machine/Enterprise Systems Architecture)
- Configured with VSE running as a guest under VM.

Specific VM operating systems are mentioned in the text when a task or DB2 Server for VM facility applies to a subset of the VM operating systems.

The following tasks are described here:

- Installation
- Migration
- Operation
- Management of resources (including security)
- Modification of facilities (including national language support)
- Installation and maintenance of Distributed Relational Database Architecture (DRDA®) facilities.

The term **database manager** refers to the DB2 Server for VM database manager, unless otherwise stated.

Organization of This Manual

- “Summary of Changes for DB2 Version 6 Release 1” on page xix lists the changes made to the product since Version 5 Release 1.
- Chapter 1, “Planning for Installation” on page 1 summarizes the software, hardware, and storage requirements for installing the database manager.
- Chapter 2, “Planning for Database Generation” on page 15 describes how to set up your initial database, including specifying parameters to define the logical and physical limits for its capacity and setting its initial DASD allocations.
- Chapter 3, “Planning for Database Migration” on page 33 explains the planning you must do before migrating a database from a previous release of the database manager to the Version 6 Release 1 level. For the actual migration steps, see the *DB2 Server for VM Program Directory*.
- Chapter 4, “Planning for Operation of the Database Manager” on page 57 explains how to choose appropriate startup parameters which will determine the operational characteristics of the application server when it is started by the DB2 Server for VM operator.
Note: Starting, operating, and stopping the application server are also discussed in the *DB2 Server for VSE & VM Operation* manual.
- Chapter 5, “Operating the Online Support for VSE Guest Sharing” on page 103 explains how to enable VSE guest users to access the application server on a VM/ESA operating system, and how to operate the online support for CICS/VSE® transactions.

Note: These subjects are also discussed in the *DB2 Server for VSE & VM Operation* manual.

- Chapter 6, “Maintaining Database Security” on page 143 discusses how to control access to the application server.
- Chapter 7, “Managing Database Storage” on page 151 explains how to manage the disk storage allocated to the database, including adding (or defining) dbspaces, defining storage pools, adding dbextents to storage pools, and managing storage pools.
- Chapter 8, “Saved Segments” on page 181 discusses using, defining and running saved segments.
- Chapter 9, “Making Backups and Recovering from Failures” on page 203 describes facilities provided for recovery from system failures and DASD failures; how to back up your database; and how to recover from different types of failures.
- Chapter 10, “Special Topics in Recovery Design” on page 237 discusses dual logging and switching log modes.
- Chapter 11, “Using the Accounting Facility” on page 261 describes the DB2 Server for VM accounting facility, which tracks how database resources are consumed by users.
- Chapter 12, “Planning and Implementing Configurations” on page 277 describes configuration topics like adding database and user machines, and configuring for different operating systems.
- Chapter 13, “Choosing a National Language and Defining Character Sets” on page 319 contains information on national language character set and coded character set identifier (CCSID) support, as well as how to provide HELP text and messages in languages supported by the database manager.
- Chapter 14, “Creating Installation Exits” on page 355 describes the types of installation exits that you can code to customize the database manager:
 - Accounting exits, to customize account information
 - Date and time exits, to create your own date or time format if the IBM-supplied formats do not fit your requirements
 - TRANSPROC exits, to carry out DBCS conversions
 - Cancel exits, to replace the product-supplied cancel function when coding your own interactive program
 - Field Procedures, to change the sorting sequence by encoding and decoding data if the standard sorting sequence does not meet your requirements.
- Chapter 15, “Using a DRDA Environment” on page 411 discusses using the database manager in a distributed environment; benefits; how to prepare DB2 Server for VM application requesters and application servers; administrative responsibilities; and using the database services utility (DBS Utility) and ISQL to access a non-DB2 Server for VM application server. Considerations for distributed databases and for choosing the PROTOCOL parameter are also discussed.
- Chapter 16, “Using TCP/IP with DB2 Server for VM” on page 435 discusses using TCP/IP to access application servers.

- Appendix A, “Virtual and Real Storage Requirements” on page 443 presents guidelines for estimating the processor requirements needed for running the database manager.
- Appendix B, “Estimating Database Storage” on page 449 contains procedures for estimating the sizes of the database directory, public dbspaces, and the ISQL dbspace.
- Appendix C, “Maximum Values” on page 469 contains the system and database maximums for the database manager.
- Appendix D, “Updating SYSTEM.SYSSTRINGS” on page 471 details how to update this catalog table to support your own CCSID conversion.
- Appendix E, “Defining Your Own Character Set” on page 475 describes how to create your own character set.
- Appendix F, “Macro List” on page 491 lists the macros identified as programming interfaces for customers by the database management system.
- Appendix G, “Service and Maintenance Execs” on page 493 lists and describes service and maintenance execs.
- Appendix H, “DRDA Considerations” on page 499 discusses what you should consider in a distributed environment.
- Appendix I, “Incompatibilities Between Releases” on page 503 describes the incompatibilities between releases.

A bibliography is provided at the back of the book.

Syntax Notation Conventions

Throughout this manual, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right and from top to bottom, following the path of the line.

The >>— symbol indicates the beginning of a statement or command.

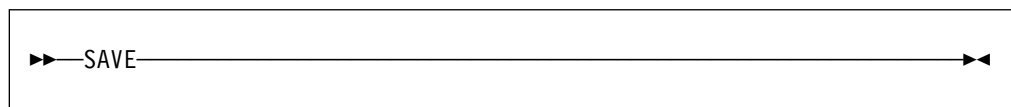
The —> symbol indicates that the statement syntax is continued on the next line.

The >— symbol indicates that a statement is continued from the previous line.

The —>< symbol indicates the end of a statement.

Diagrams of syntactical units that are not complete statements start with the >— symbol and end with the —> symbol.

- Some SQL statements, Interactive SQL (ISQL) commands, or database services utility (DBS Utility) commands can stand alone. For example:



Others must be followed by one or more keywords or variables. For example:

▶▶—SET AUTOCOMMIT OFF—◀◀

- Keywords may have parameters associated with them which represent user-supplied names or values. These names or values can be specified as either constants or as user-defined variables called *host_variables* (*host_variables* can only be used in programs).

▶▶—DROP SYNONYM—*synonym*—◀◀

- Keywords appear in either uppercase (for example, SAVE) or mixed case (for example, CHARACTER). All uppercase characters in keywords must be present; you can omit those in lowercase.
- Parameters appear in lowercase and in italics (for example, *synonym*).
- If such symbols as punctuation marks, parentheses, or arithmetic operators are shown, you must use them as indicated by the syntax diagram.
- All items (parameters and keywords) must be separated by one or more blanks.
- Required items appear on the same horizontal line (the main path). For example, the parameter *integer* is a required item in the following command:

▶▶—SHOW DBSPACE—*integer*—◀◀

This command might appear as:

```
SHOW DBSPACE 1
```

- Optional items appear below the main path. For example:

▶▶—CREATE—
 └─UNIQUE─┘ INDEX—◀◀

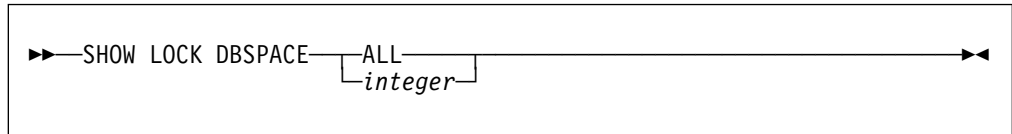
This statement could appear as either:

```
CREATE INDEX
```

or

```
CREATE UNIQUE INDEX
```

- If you can choose from two or more items, they appear vertically in a stack. If you must choose one of the items, one item appears on the main path. For example:



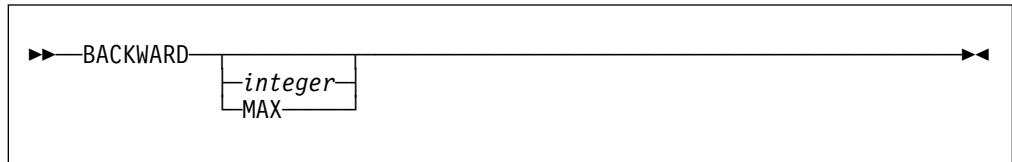
Here, the command could be either:

SHOW LOCK DBSPACE ALL

or

SHOW LOCK DBSPACE 1

If choosing one of the items is optional, the entire stack appears below the main path. For example:



Here, the command could be:

BACKWARD

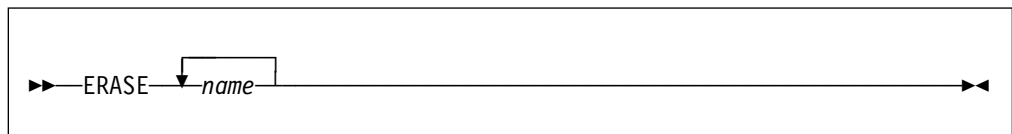
or

BACKWARD 2

or

BACKWARD MAX

- The repeat symbol indicates that an item can be repeated. For example:



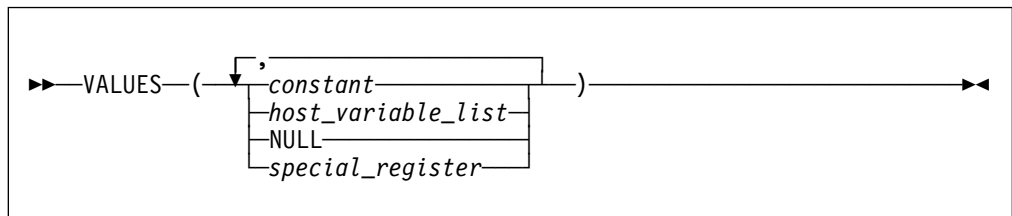
This statement could appear as:

ERASE NAME1

or

ERASE NAME1 NAME2

A repeat symbol above a stack indicates that you can make more than one choice from the stacked items, or repeat a choice. For example:



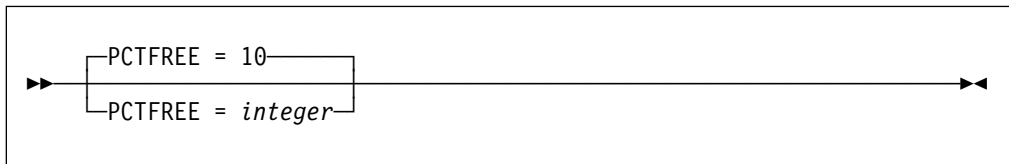
- If an item is above the main line, it represents a default, which means that it will be used if no other item is specified. In the following example, the ASC

keyword appears above the line in a stack with DESC. If neither of these values is specified, the command would be processed with option ASC.

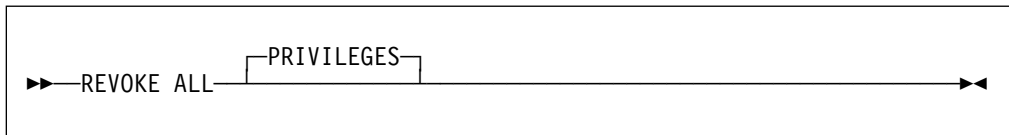


- When an optional keyword is followed on the same path by an optional default parameter, the default parameter is assumed if the keyword is not entered. However, if this keyword is entered, one of its associated optional parameters must also be specified.

In the following example, if you enter the optional keyword PCTFREE =, you also have to specify one of its associated optional parameters. If you do not enter PCTFREE =, the database manager will set it to the default value of 10.

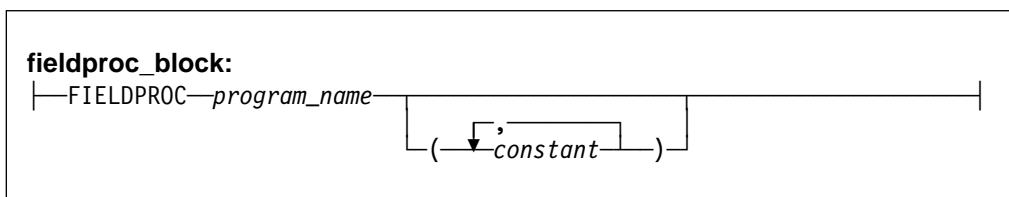
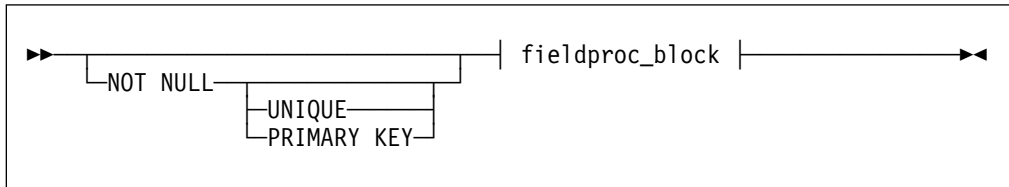


- Words that are only used for readability and have no effect on the execution of the statement are shown as a single uppercase default. For example:



Here, specifying either REVOKE ALL or REVOKE ALL PRIVILEGES means the same thing.

- Sometimes a single parameter represents a fragment of syntax that is expanded below. In the following example, **fieldproc_block** is such a fragment and it is expanded following the syntax diagram containing it.



SQL Reserved Words

The following words are reserved in the SQL language. They cannot be used in SQL statements except for their defined meaning in the SQL syntax or as host variables, preceded by a colon.

In particular, they cannot be used as names for tables, indexes, columns, views, or dbspaces unless they are enclosed in double quotation marks ("").

ACQUIRE	GRANT	RESOURCE
ADD	GRAPHIC	REVOKE
ALL	GROUP	ROLLBACK
ALTER		ROW
AND	HAVING	RUN
ANY		
AS	IDENTIFIED	SCHEDULE
ASC	IN	SELECT
AVG	INDEX	SET
	INSERT	SHARE
BETWEEN	INTO	SOME
BY	IS	STATISTICS
		STORPOOL
CHAR	LIKE	SUM
CHARACTER	LOCK	SYNONYM
COLUMN	LONG	
COMMENT		TABLE
COMMIT	MAX	TO
CONCAT	MIN	
CONNECT	MODE	UNION
COUNT		UNIQUE
CREATE	NAMED	UPDATE
CURRENT	NHEADER	USER
	NOT	
DBA	NULL	VALUES
DBSPACE		VIEW
DELETE	OF	
DESC	ON	WHERE
DISTINCT	OPTION	WITH
DOUBLE	OR	WORK
DROP	ORDER	
EXCLUSIVE	PACKAGE	
EXECUTE	PAGE	
EXISTS	PAGES	
EXPLAIN	PCTFREE	
	PCTINDEX	
FIELDPROC	PRIVATE	
FOR	PRIVILEGES	
FROM	PROGRAM	
	PUBLIC	

Summary of Changes for DB2 Version 6 Release 1

This is a summary of the technical changes to the DB2 Server for VSE & VM Version 6 Release 1 database management system. All manuals are affected by some or all of the changes discussed here. This summary does not list incompatibilities between releases of the DB2 Server for VSE & VM product; see either the *DB2 Server for VSE & VM SQL Reference*, *DB2 Server for VM System Administration*, or the *DB2 Server for VSE System Administration* manuals for a discussion of incompatibilities. Version 6 Release 1 of the DB2 Server for VSE & VM database management system is intended to run on the Virtual Machine/Enterprise Systems Architecture (VM/ESA®) Version 2 Release 2 or later environment and on the Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA™) Version 2 Release 2 or later environment.

Enhancements, New Functions, and New Capabilities

DRDA® RUOW Application Requestor for VSE (Online)

DRDA Remote Unit of Work Application Requestor provides read and update capability in one location in a single unit of work.

This support provides CICS/VSE® online application programs with the ability to execute SQL statements to access and manipulate data managed by any remote application server that implements the DRDA architecture. Online application programs that access remote application servers need to be preprocessed to create a bind file and then bound (using CBND) to the remote application server. Online application programs that access a local application server are preprocessed as in previous releases.

See the following DB2 Server for VSE & VM manuals for further information:

- *DB2 Server for VSE System Administration*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE Database Administration*
- *DB2 Server for VSE Application Programming*
- *DB2 Server for VSE Installation*

Stored Procedures

The ability to use stored procedures provides distributed solutions that let more people access data faster.

A stored procedure is a user-written application program compiled and stored at the server. When the database is running in multiple user mode, local applications or remote DRDA applications can invoke the stored procedure. SQL statements are local to the server and issued by a stored procedure so they do not incur the high network costs of distributed statements. Instead, a single network send and receive operation is used to invoke a series of SQL statements contained in a stored procedure.

See the following DB2 Server for VSE & VM manuals for further information:

- *DB2 Server for VM System Administration*
- *DB2 Server for VM Database Administration*
- *DB2 Server for VSE & VM SQL Reference*
- *DB2 Server for VSE & VM Operation*

TCP/IP Support for DB2 Server for VM

TCP/IP support allows:

- VM applications to use SQLDS-private protocol to connect to VM databases over TCP/IP.
- VM applications to use DRDA protocol to connect to DB2 family databases (and any other database that supports DRDA connections) over TCP/IP.
- non-VM applications to use DRDA-protocol to access VM database over TCP/IP.

TCP/IP support for DB2 Server for VM integrated with the DB2 Server for VM application server means a system easier to configure and maintain.

The database manager will optionally secure TCP/IP connections using any external security manager that supports the RACROUTE interface.

New Code Page and Euro Symbol Code Page Support

The following CCSIDs are now supported:

- 1112: Latvian/Lithuanian
- 1122: Estonian
- 1123: Ukrainian
- 1130: Vietnamese
- 1132: Lao
- 1148: E-International
- 1140: E-English
- 1141: E-German
- 1144: E-Italian
- 1146: E-UK-English
- 1147: E-French

Additional support has been added for conversions from Unicode (UCS-2) to host CCSIDs.

For a complete list of CCSIDs supported refer to the *DB2 Server for VM System Administration* and *DB2 Server for VSE System Administration* manuals.

DataPropagator™ Capture

DataPropagator Capture is part of the DB2 Family of DataPropagator products. DataPropagator Capture is updated for Version 6 Release 1 compatibility.

QMF for VM, QMF for VSE, and QMF for Windows®

IBM Query Management Facility (QMF™) is now an separately priced feature of DB2 Server for VSE & VM. QMF is a tightly integrated, powerful, and reliable tool that performs query and reporting for IBM's DB2 relational database Management System Family. It offers an easy-to-learn, interactive interface. Users with little or no data processing experience can easily retrieve, create, update, insert, or delete data that is stored in DB2.

QMF offers a total solution that includes accessing large amounts of data and sharing central repositories of queries and enterprise reports. It also allows you to implement tightly-controlled, distributed, or client-server solutions. In addition, you can use QMF to publish reports to the World Wide Web that you can view with your favorite web browser.

Using QMF, users can access a wide variety of data sources, including operational or warehouse data from many platforms: DB2 for VSE, VM, OS/390® and Windows. Via IBM Data Joiner, you can access non-relational data, such as IMS™ and VSAM, as well as data from other vendor platforms.

RDS Above the Line

The RDS component will load and execute above the 16 megabyte line. This support frees up approximately 1.5 megabytes of storage below the 16 megabyte line (or approximately 2.5 megabytes, if DRDA is installed) when compared to Version 5 Release 1. No installation or migration changes are required for this support to be used (except for the definition of VM Shared Segments and for users who execute the database server with AMODE(24)). If sufficient storage is available, the RDS component will be automatically loaded above the 16 megabyte line. When using VM Shared Segments, the RDS Segment should be defined above the 16 megabyte line.

VM users who wish to run the database server in 24-bit addressing mode (i.e. use the AMODE(24) parameter) **must** use a virtual storage size no greater than 16 megabytes. See the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* for release to release incompatibility information.

Combining of NLS Feature Installation Tapes with Base Product Installation Tape

All available NLS features for DB2 Server for VSE, DB2 Server for VM, Control Center for VSE and REXX SQL for VM have been combined with the respective base product installation tape. Customers interested in an NLS feature language will no longer need to order an additional NLS feature tape because all NLS languages will be available to all customers. In all cases, the default language as shipped is American English. The installation and migration processes have been changed to allow you to choose the default language. Refer to the *DB2 Server for VM Program Directory*, *DB2 Server for VSE Installation*, *DB2 for VSE Control Center Installation and Operations Guide*, and *DB2 REXX SQL for VM/ESA Installation* for the details of how these changes affect the installation process and how you can choose to have a different default language.

Control Center Feature

DB2 Server for VSE & VM Version 6 Release 1 enhances the new Control Center feature as follows:

For both VM/ESA and VSE/ESA:

- Access to the Query Management Facility (QMF)

For VM/ESA:

- Compatibility with DB2 Server for VM Version 6 Release 1 initialization parameters and operator commands
- Shared File System Support (SFS) in a VM/ESA environment
- CA-DYNAM/T Interface Support in a VM/ESA environment
- Data Restore Incremental Backup Support in a VM/ESA environment

For VSE/ESA:

- Control Center code installation on any library
- Ability to use while viewing a list of tables online
- Ability to create, reorganize, unload, reload, move and copy tables in batch mode
- Ability to update table statistics in batch mode
- Ability to drop tables online

Data Restore Feature

The Data Restore feature provides archiving and recovery functions in addition to those provided in DB2 for VSE & VM. Data Restore is enhanced in Version 6 Release 1 with incremental database archiving support. The support allows you to archive only the areas of the database that have been updated since the last database archive, instead of having to archive the entire database. This can provide significant savings for customers with large databases which are updated infrequently, or where only a small fraction of the database is updated frequently.

DB2 REXX SQL Feature

The DB2 REXX SQL feature provides a REXX interface for VM customers to allow SQL calls to be executed from REXX programs. The DB2 REXX SQL feature is updated for Version 6 Release 1 compatibility.

Reliability, Availability, and Serviceability Improvements

Migration Considerations

Migration is supported from SQL/DS™ Version 3 and DB2 Server for VSE & VM Version 5. Migration from SQL/DS Version 2 Release 2 or earlier releases is not supported. Refer to the *DB2 Server for VM System Administration* or *DB2 Server for VSE System Administration* manual for migration considerations.

Library Enhancements

Some general library enhancements include:

- The following books have been removed from the library:
 - *DB2 Server for VM Operation*
 - *DB2 Server for VSE Operation*
 - *DB2 Server for VM Interactive SQL Guide and Reference*
 - *DB2 Server for VSE Interactive SQL Guide and Reference*
 - *DB2 Server for VM Database Services Utility*
 - *DB2 Server for VSE Database Services Utility*
- The following books have been added to the library:
 - *DB2 Server for VSE & VM Operation*
 - *DB2 Server for VSE & VM Interactive SQL Guide and Reference*
 - *DB2 Server for VSE & VM Database Services Utility*

Refer to the new *DB2 Server for VSE & VM Overview* for a better understanding of the benefits DB2 Server for VSE & VM can provide.

Chapter 1. Planning for Installation

This chapter discusses the tasks that need to be done before you begin to install the DB2 Server for VM database manager. Details on how to perform installation can be found in the *DB2 Server for VM Program Directory*.

Operating System Overview

The database manager runs on a VM/ESA Version 2 Release 2 or later operating system. If you are running VSE as a guest operating system under VM, VSE users and applications can access DB2 Server for VM servers. This feature is called VSE Guest Sharing.

If DRDA code is installed, DB2 Server for VM requesters can use the DRDA protocol to access servers on other platforms, and requesters on other platforms can use the DRDA protocol to access a DB2 Server for VM database. For more information, see Chapter 15, "Using a DRDA Environment" on page 411.

Note that this product does not support mixed levels of CP and CMS.

Virtual Machine Overview

This section provides an overview of the virtual machines required by the database manager.

You need one or more of the following virtual machines for installation and subsequent use:

1. *MAINT*. This machine already exists in your VM system.
2. *Installation User ID*. You need it to use VMSES/E to install, service, and migrate the database manager.
3. *Database machine*. A database machine is a virtual machine in which the database manager code runs. There can be more than one database machine.

The database machine owns a database. It provides all database management services for a database. The database machine processes SQL requests from users, and returns the results to the users.

Note: The word "own," in this context, describes the association of the database machine's user ID as the owner of the minidisks that contain the database. Users who want to link to and access minidisks owned by another user must be authorized by the owner or the system administrator.

The VM database machines can be defined as either a LOCAL resource, which restricts access to users on the same processor, or a GLOBAL resource, which allows access to users on other processors. For more information on defining databases, see "VM Directory Control Statements" on page 146.

4. *Service machine*. A service machine is required by any processor that does not have its own database machine, and has users who want to access data in either of the following:

- A DB2 Server for VM database in a Transparent Services Access Facility (TSAF) collection
- A relational database residing in an SNA network that uses remote unit of work or distributed unit of work support.

DRDA Remote Unit of Work (RUOW) support was introduced in SQL/DS Version 3 Release 3. DRDA Distributed Unit of Work (DUOW) server support is introduced in DB2 Server for VM Version 5 Release 1. For more information on the DRDA environment, see Chapter 15, "Using a DRDA Environment" on page 411.

The service machine provides essential support to users by allowing access to a DB2 Server for VM production minidisk. The production minidisk contains files required by the users. For example, the SQLINIT EXEC files that enable ISQL (Interactive Structured Query Language), the DBS utility, and the preprocessors are located on this minidisk.

For information on installing the service machine and the files that it uses, see the *DB2 Server for VM Program Directory*.

5. *User machine*. A user machine is a virtual machine that has read access to the database machine production minidisk.

Components of the Relational Database Management System

Figure 1 on page 3 depicts a typical configuration with one database and two users.

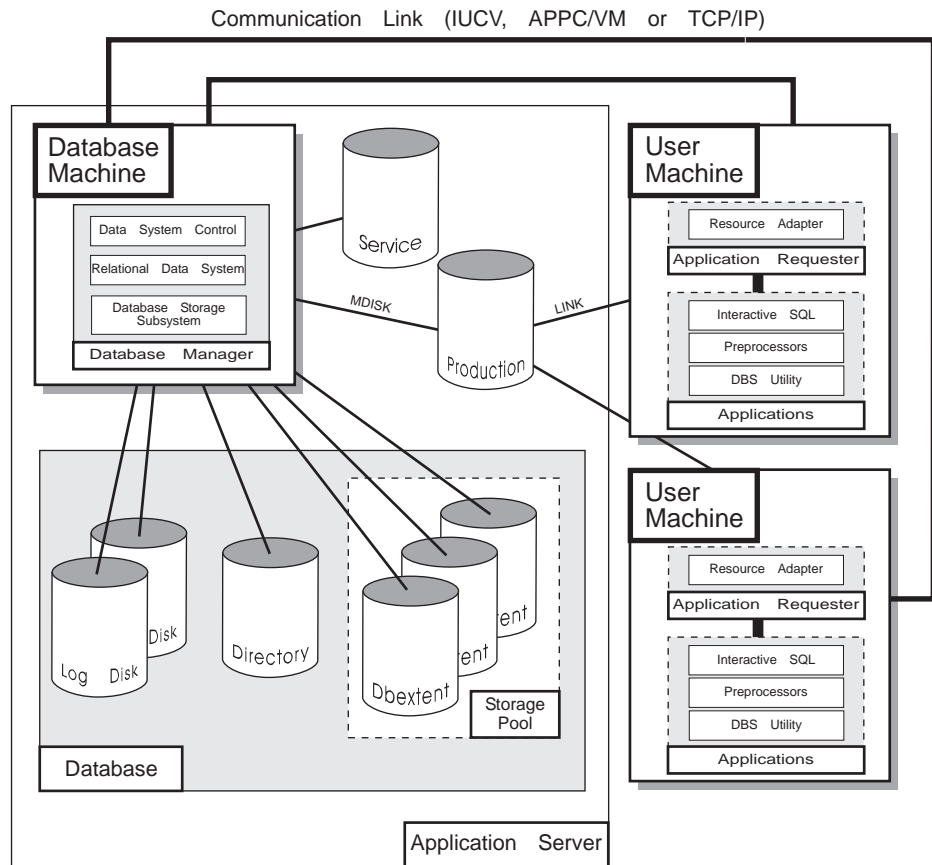


Figure 1. Basic Components of the RDBMS in VM/ESA

The **database** is composed of :

- A collection of data contained in one or more *storage pools*, each of which in turn is composed of one or more *database extents (dbextents)*. A dbextent is a VM minidisk.
- A *directory* that identifies data locations in the storage pools. There is only one directory per database.
- A *log* that contains a record of operations performed on the database. A database can have either one or two logs.

The **database manager** is the program that provides access to the data in the database. It is loaded into the database virtual machine from the production disk.

The **application server** is the facility that responds to requests for information from and updates to the database. It is composed of the database and the database manager.

The **application requester** is the facility that transforms a request from an application into a form suitable for communication with an application server.

Software Requirements

The database manager for VM requires an environment provided by IBM Virtual Machine/Enterprise Systems Architecture (VM/ESA operating system), Version 2 Release 2 or later either by itself, or as the base of any VM package. Depending on your intended use of the database manager, you may need other licensed program products, as follows:

- For VSE guest sharing, VSE is required.
- For remote printing by ISQL you need the remote spooling communications subsystem (RSCS) Version 2 Release 2 or later.
- To develop DB2 Server for VM application programs, you can use one or more of the following compilers:
 - A PL/I compiler
 - A COBOL compiler
 - A VS FORTRAN compiler
 - A C/370 Compiler
 - An Assembler.
- The database manager supports some of the enhancements of COBOL II Release 3. You can take advantage of these enhancements if you have a COBOL II Release 3 compiler. For information on using COBOL II Release 3 with the database manager, see the *DB2 Server for VM Application Programming* manual.
- To use double-byte character set (DBCS) characters in application programs, you need the following compilers:
 - COBOL II Release 2 compiler
For COBOL II Release 2 programs, SQL identifiers, SQL host variables, and SQL labels with DBCS characters can be used in SQL statements. The COBOL Kanji preprocessor is not required.
 - PL/I compiler
For PL/I Release 2.1 programs, SQL identifiers, SQL host variables, and SQL labels with DBCS characters can be used in SQL statements.
 - VS FORTRAN Version 2 Release 3 compiler
VS FORTRAN Version 2 Release 3 programs support DBCS symbolic names and DBCS characters in character constants.
 - C/370 compiler
For C/370 programs, SQL identifiers, SQL host variables, and SQL labels with DBCS characters can be used in SQL statements.
 - Assembler compiler
DBCS variables and constants are not supported in Assembler programs. You can still use DBCS characters in dynamically defined SQL statements.
- If you want to use REXX, you need RXSQL.
When using RXSQL, you cannot use DBCS characters in cursor names and statement names.
- To provide remote unit of work access between application requesters and application servers in an SNA network, you need VTAM* Version 3 Release 2

or later. If you want to use either partner LU verification or SECURITY=SAME conversations that are routed through AVS, you must have VTAM Version 3 Release 3 and RACF* Version 1 Release 9, or an equivalent security manager product.

For more information about remote unit of work in a DRDA environment, see the *Distributed Relational Database Connectivity Guide* manual.

- To archive the database using user facilities, consider using the DASD Dump Restore (DDR) utility included with your VM operating system.

Note: All references to above programs apply to equivalent non-IBM products.

Virtual Storage Requirements

All DB2 Server for VM operations are serviced by *database machines*. A database machine is a virtual machine in which the system code (the components of the database manager) runs.

Database Machine Size

The amount of virtual storage required by the database machine depends on several factors. The dominant ones are the sizes of the buffer pools (used for the directory and the data), the number of concurrent users to be supported, the complexity of the SQL requests, and the definition of the database. If only the base code has been installed, 8 megabytes are required for the database machine; if the DRDA code has been installed as well, 10 megabytes are required.

If you have coded any date or time exit routines, the size of these routines must be added to the minimum virtual storage. For more information, see “Defining Your Own Datetime Format” on page 368. ISQL cannot be run in the database machine.

If you plan to run the database manager in single user mode, add the size of the application being run, and the resource adapter. See Table 1 on page 6 for more information on virtual storage requirements. For a description of resource adapter, see “How the ARIEXIT Module Works” on page 356.

The database manager (as a default) runs in the user free storage area. (The entire user program area is used for user free storage when the database manager is running.)

For detailed formulas for calculating virtual storage requirements, see Appendix A, “Virtual and Real Storage Requirements” on page 443.

Service Machine Size

If CMS is defined as a saved segment, a 1-megabyte machine size is recommended; if not, ensure that you define enough virtual storage for the service machine to run CMS. For information on saved segments and how they are defined see “Defining Saved Segments” on page 187.

The service machine does not perform any processing. Its purpose is to own the database manager minidisks so that SQL users can access the IBM-supplied programs, such as ISQL, the DBS utility, and the preprocessors.

User Machine Size

Table 1 shows the recommended increase to virtual storage requirements for a user machine. A 6-megabyte machine size is recommended.

DB2 Server for VM Component	Recommended Minimum Virtual Storage Increase for a User Virtual Machine (in Kilobytes)
Resource adapter	242 (if base code installed) 2048 (if DRDA code installed)
ISQL	320 + resource adapter = 562 (base) 320 + resource adapter = 2368 (DRDA)
DBS utility	220 + resource adapter = 462 (base) 220 + resource adapter = 2268 (DRDA)
Any DB2 Server for VM preprocessor (PL/I, Assembler, COBOL, FORTRAN, or C)	222 + resource adapter = 464 (base) 222 + resource adapter = 2270 (DRDA)
User program	User application size + resource adapter

To these minimum amounts, add the following amount of storage as needed:

- The values in Table 1 assume that national language message repositories are in saved segments. For each such repository not in a saved segment, add 250 kilobytes. If you use CMS work units, also add 5 kilobytes for the second and each subsequent CMS work unit used by the application.
- Add the size of CMS to these amounts if it is not defined as a saved segment.
- For user programs that use the SQLDS protocol, if the fetch and insert blocking option is in effect, add 8 kilobytes for each currently open cursor that qualifies for blocking.
- For user programs that use the DRDA protocol, add the query block size for each open cursor. (For information on the SQLINIT query block size parameter, see the *DB2 Server for VM Database Administration* manual.)
- If you have coded an exit, such as an accounting or a cancel exit, add the size of your routine.

Hardware Requirements

Hardware requirements include real storage, DASD space, tape, and display terminals.

Real Storage Requirements

The database manager itself does not require any real storage. However, if more real storage is available, there is less paging, thus improving performance.

The VSE guest sharing facility requires 40 kilobytes of real storage for each database communication link.

DASD Space Requirements

The DASD space requirements for the virtual machines are discussed below.

Minidisks Required for the Installation User ID Machine

You no longer install and service DB2 Server for VM strictly using the MAINT user ID. You should use the user ID, 5648A70S. You can change this user ID, however, by creating a PPF override. See the *DB2 Server for VM Program Directory* for more information.

See the *DB2 Server for VM Program Directory* for the recommended DASD sizes for the installation user ID machine and initial installation, migration, and service instructions.

Minidisks Required for a Database Machine

A database machine requires two kinds of minidisks: system minidisks and database minidisks.

System Minidisks: A database machine must have read/write access to its own work minidisk (A-disk). In addition, it must be able to access the DB2 Server for VM production and service minidisks. Collectively, these three minidisks are referred to as the *system minidisks*. The system minidisks can be optionally installed in shared file system (SFS) directories with default names of VMSYS:SQLMACH, VMSYS:SQLMACH.SQL.PRODUCTION and VMSYS:SQLMACH.SQL.SERVICE. From now on, any reference to the service or production minidisk can be replaced by these directories.

The work minidisk is required because the database manager does various operations that require space temporarily.

The production minidisk contains IBM-supplied EXECs and programs that are required for day-to-day use of the database manager. The production minidisk defines an **entire** DB2 Server for VM environment, and contains all the CMS files that enable database machines to access databases. It also contains CMS files that allow users to access a database with a given database machine. The CMS files determine the default application server and thus the database a user can access. Users can access other database machines and thus other databases by database switching.

The DB2 Server for VM LOADLIB resides on the production minidisk. Every virtual machine must have read access to the production minidisk in order to access the database manager.

The service minidisk also contains IBM-supplied EXECs and programs, but it needs to be accessed only during installation, database generation, migration, maintenance or system administration activities.

Usually, all database machines use the same production and service minidisks. Thus, they can be defined once for the entire installation. You can define more than one production minidisk as your installation grows. Multiple production minidisks are convenient when you have many database machines that often perform administrative tasks. If you define a second production minidisk, you create a second DB2 Server for VM environment. This environment has its own users, its own database machines, and its own databases. It is independent of any other DB2 Server for VM environment that is defined by any other production minidisk. More

specific information is in Chapter 12, “Planning and Implementing Configurations” on page 277. In planning your initial installation, assume there will be only one production minidisk and one service minidisk.

Initially, there is only one database machine, SQLMACH. Thus, for installation, you need to be concerned only with the size of the work minidisk for that initial database machine. The installation process makes SQLMACH the owner of the production and service minidisks.

See the *DB2 Server for VM Program Directory* for the recommended database machine DASD sizes for the service minidisks and production minidisks.

A service minidisk must contain only IBM-supplied files: it must not contain any user-created files. The IBM-supplied service minidisk files are documented in the *DB2 Server for VM Program Directory*.

The production minidisk may contain user files, but it must contain all the IBM-supplied files. The IBM-supplied production minidisk files are documented in the *DB2 Server for VM Program Directory*. The space allocations shown for the production minidisk reflect the requirements for the IBM-supplied files plus approximately 30% free space.

The service minidisk allocation for non-English versions of the HELP text is described in the *program directory* supplied with the non-English HELP text distribution tape. The allocations documented in the *DB2 Server for VM Program Directory* include space for the English version of the HELP text.

The service minidisk is referred to as the SQLMACH 193 minidisk, but can be defined with any valid user ID and virtual address. Similarly, although the production minidisk is referred to as the SQLMACH 195 minidisk, you can use any valid user ID and virtual device address. The same virtual machine must own both the service and production minidisks. When migrating from a previous release of the database manager, another user ID and virtual device address can be used for the production minidisk for testing. If you have not used a previous release of the database manager, you should use the user ID SQLMACH and the 193 and 195 virtual device addresses as described in the *DB2 Server for VM Program Directory*.

Database Minidisks: Minidisk requirements for a database machine vary based on the number and size of databases defined on it. Each database has a minimum minidisk storage requirement. A database requires a minimum of three VM minidisks, but a typical database has several more. The minidisk requirements are summarized below:

- A directory minidisk to hold internal control information for the database.
- Either one or two log minidisks, to hold recovery information. Only one is required, but defining two log minidisks on separate volumes is recommended, to protect against I/O errors on access to the log information.
- Database extents (dbextents) to hold the user data of the database. It is possible to have only one dbextent, but a typical database has several.

The directory, log and database extents minidisks cannot be in CMS shared file system directories.

The directory and log minidisks are discussed further in Chapter 2, “Planning for Database Generation” on page 15. Dbextent minidisks are discussed in greater detail in Chapter 7, “Managing Database Storage” on page 151.

The Starter Database

The ARISDBG MACRO, which comes with this product, contains IBM-supplied specifications for generating a starter database. This database consists of one directory minidisk, one log minidisk, and one data minidisk. You can later add more dbextents, up to a logical maximum size of about 4.6 gigabytes, using the information in “Adding Dbextents to a Storage Pool” on page 166.

You should generate the starter database at the time of initial installation and experiment with it in order to familiarize yourself with the database manager. You may then keep it as your production database. However, as your needs grow, you may find it necessary to transfer its contents to another database, which can be a major undertaking. Thus, once you are familiar with how it works, it is best to discard the starter database and generate your own database by following the guidelines in Chapter 2, “Planning for Database Generation” on page 15.

The initial physical size of the starter database is predefined and will be about the same on all IBM storage devices. See the *DB2 Server for VM Program Directory* for the recommended DASD sizes for the starter database.

This starter database must be able to fit in a single dbextent. If you do not have enough DASD, you will not be able to use the IBM-supplied specifications, and will have to generate your own database at the time of installation. If you want to define the equivalent of the starter database on the devices, you must define multiple dbextents on multiple volumes.

If you are migrating from a previous release of the database manager, you already have at least one database, so generating the starter database is optional. The advantage of doing so is that you can use it as a test database to verify your installation, but the disadvantages are the work involved and the necessary DASD allocations. Thus to deal with migration needs, the database manager provides allocations for generating a starter database that is large enough to hold the initial database components (for example, HELP text, catalog tables, and FORTRAN packages), but not much else. The *DB2 Server for VM Program Directory* also shows the minidisk sizes for a minimum starter database.

Minidisks Required for a Service Machine

The service machine must have read/write access to its own work minidisk (A-disk). In addition, it must be able to access the production and service minidisks. For more information see “System Minidisks” on page 7.

If you only have a service machine on a processor and intend to access a database manager (defined as a global resource) in another processor, the code that is installed on your local processor is known as the service machine. If you install the service machine, you do not need a database machine. See the *DB2 Server for VM Program Directory* for the recommended DASD sizes for the service machine.

Minidisks Required for a User Machine

During installation, it is recommended that one virtual machine be defined as a user machine. A user machine also requires a 191 minidisk (A-disk, formatted at the 1024 byte block size with free space equivalent to at least 3 cylinders of an IBM 3380 storage device). The user machine 191 disk can optionally be installed in a CMS shared file system directory. See the *DB2 Server for VM Program Directory* for the recommended DASD sizes for the user machine. After initial installation, you will probably want to define many user machines.

Tape Requirements

One tape drive is required for installation. Depending on the DB2 Server for VM facilities you use, you may need tape drives after installation. Tape processing can be used for the following activities:

- Database archive and log archive processing (both creating the archive and restoring the database from the archive) to support recovery from DASD failures
- Unloading and reloading data into the database using the DBS utility
- Holding the output of the trace facility
- Holding the output of the accounting facility

For all of these facilities except archiving, you can use DASD instead of tape.

Also, with the exception of accounting output, the database manager does not require the continuous use of any tape drive: tape mounts are requested when needed. If you are using tape drives, you should have at least two to cover all your needs.

The database manager supports all tape drives that are supported by the operating system.

Display Terminal Requirements

A variety of display terminals are supported, including the larger screen sizes offered by some models of the 3278 and 3279 (or equivalent) devices. Since the database manager relies on CP (control program) and CMS to provide terminal support for DB2 Server for VSE online applications, the terminal must be one that is supported by CMS.

You can direct ISQL-printed output to any printer supported by the Remote Spooling Communications Subsystem (RSCS). Use CP SPOOL and TAG commands to change the routing of the print output.

Note: To display and print DBCS characters (for example, Japanese HELP text), a DBCS terminal and printer (for example, the IBM 5550 terminal) are required.

Considerations When Defining a Database Machine and Generating a Database

Read this section before carrying out any activities.

Considerations When Adding Directory Control Statements

VM directory control statements are describe on 146. You add them to your database machine to:

- Define its virtual storage
- Enable communications between it and the user machines or gateways.
- Provide links to the service, production, and database minidisks. (The database machine's PROFILE EXEC must be updated so that these minidisks or CMS shared file directories can be accessed.)

Considerations When Loading IBM-Supplied Files

Install all the IBM-supplied files into the production and service minidisks. This loads the full product version to support both a database and your users.

Considerations When Generating a Database

You generate a database by specifying parameters to define its maximums. It is recommended that the first time you do so, you use the IBM-supplied set of predefined parameters called the *starter database specifications*. (See the *DB2 Server for VM Program Directory* for instructions on how to generate a starter database.) Once you are familiar with how the starter database works, you should delete it and generate your own. Refer to Chapter 2, "Planning for Database Generation" on page 15.

Considerations When Defining a Service Machine

Read this section if you plan to define a service machine.

Updating the Service Machine VM Directory

Add VM directory control statements for the service virtual machine to:

- Define its virtual storage
- Provide links to the service and production minidisks.

More information is provided in Chapter 12, "Planning and Implementing Configurations" on page 277.

Considerations When Loading IBM-supplied Files

Since this is a service machine, you should not generate a database. However, because all the DB2 Server for VM files are already loaded where you installed the service machine, you can generate a database in the future. For information on generating a database on a processor that has a service machine, see "Converting a Service Machine to a Database Machine" on page 53.

Defining User Machines

Carry out the following procedure for each user machine that you define:

1. Add VM directory control statements to:
 - a. Define the virtual storage for the user machine
 - b. Provide a link to a database machine production minidisk
2. Add (optionally) IUCV statements for communications between the user machine and a specified database machine or gateway machine.
3. Update the user machine's PROFILE EXEC so that it can access the production minidisk.

For information on performing these steps, see "Defining Additional User Machines" on page 294.

Defining Saved Segments

You can load saved segments any time after installation or migration of the database manager using VMSES/E, which uses the ARISAVES EXEC.

The instructions for defining saved segments are explained in Chapter 8, "Saved Segments" on page 181.

Setting Up the CMS Communications Directory

You must define a CMS communications directory when one of the following is true:

- You access a remote application server through the VTAM product
- The resid and server name are not the same
- You use SECURITY=PGM
- The database name is longer than 8 bytes
- You access a database using TCP/IP.

The CMS communications directory provides SNA address resolution for the application servers. If this file does not exist or does not contain an entry, or if you issue the COMDIR OFF BOTH command, then the following assumptions are made: that the application server name is the same as the resid, the application server is within the same TSAF collection as the application requester, and SECURITY=SAME.

Any type of abend in the user machine (whether it is an application or initiated by the application requester code) can cause the CMS communications directory in virtual storage to be unloaded automatically. If there is a user-level directory, reload it by using the SET COMDIR RELOAD USER command. If there is a system-level directory, reload it by using the SET COMDIR RELOAD SYSTEM command.

Issue the QUERY COMDIR command: if the directory is unusable, this command returns an UNLOADED indicator. In this case, the communications directory is reloaded for you.

Note: DB2 Server for VM makes use of the CMS NAMEFIND command when resolving CMS communications directory nicknames from database names.

The NAMEFIND command should be issued within PROFILE EXEC or SYSPROF EXEC after the SET COMDIR command to prevent virtual storage fragmentation.

For more information on the CMS communications directory, see the *VM/ESA: Connectivity Planning, Administration, and Operation* manual. If you intend to access non-DB2 Server for VM application servers, also refer to the *Distributed Relational Database Connectivity Guide* manual.

Updating the SNA NETID File

If your host machine is part of an SNA network, you must update the SNA NETID file to include your NETID (network identifier). The NETID is used in the generation of the LU 6.2 LUWID (logical unit of work identifier), which is necessary for distributed processing. With distributed processing, a DB2 Server for VM application server can receive requests from both DB2 Server for VM and non-DB2 Server for VM application requesters, and DB2 Server for VM application requesters can access non-DB2 Server for VM application servers.

You can create or change the SNA NETID file using an editor. The NETID that you specify should be that of the SNA network of which your host system is a part. It must be from one to eight characters long, and must begin in column 1 of the SNA NETID file. Your VTAM administrator can provide the NETID that you should use. (The default NETID supplied in the SNA NETID file is SNANETID. If you do not specify a valid NETID, SNANETID will be used.) The new NETID is used in the next database startup.

If you want to ensure the uniqueness of your NETID, ask your IBM representative about the IBM SNA Network Registry service.

Chapter 2. Planning for Database Generation

As described in “The Starter Database” on page 9, when you first install the database manager you should generate an initial database using the IBM-supplied specifications. This eases installation, and enables you to gain experience with the system.

However, once you know how to work with this database, you will probably want to discard it and create several databases that are tailored to your own needs. This chapter describes the parameters that are set at the time of database generation, and presents some general design considerations.

If you are migrating from an earlier version of the database manager, then instead of reading this chapter go to Chapter 3, “Planning for Database Migration” on page 33.

The database-generation process does not require definition of any data specifics; it merely establishes the potential capacity of the database. Some of the capacity-planning decisions require knowledge of the data and application requirements of your users. For example, to estimate how big the database will become, you need to know the potential number of tables that will be stored, and the storage requirements of those tables. To obtain this information, consult with the person responsible for the data and application requirements for the database. Also refer to the *DB2 Server for VM Database Administration* manual.

Database Generation Parameters

Planning for the generation of a database entails establishing logical and physical limits for its capacity, and setting its initial DASD allocations.

The parameters that you must establish at this time are summarized in Table 2. This figure also shows the IBM-provided values used for the starter database.

Note: The parameters that have a Yes entry in the **Fixed** column must be established during generation of the database, and cannot be changed for the lifetime of the database. Also note that some parameters are established by the VM directory MDISK control statements, whereas others are established by input to an IBM-supplied EXEC called SQLDBGEN.

Following the figure is a discussion of how to set these parameters, and of the issues to consider when setting them.

Parameter	Default	Minimum	Maximum	Starter Database	Fixed	Set by
Database directory size	None	1 cylinder	1 volume	34 cylinders	No	MDISK
Log data set (or data sets) -Size (each) -Number	None None	1 cylinder 1	524,287 4Kb pages 2 volumes	8 cylinders 1	No	MDISK
Maximum number of storage pools (MAXPOOLS)	32	1	999	256	Yes	SQLDBGEN

Table 2 (Page 2 of 2). Database Parameters Set at Database Generation Time

Parameter	Default	Minimum	Maximum	Starter Database	Fixed	Set by
Maximum number of dbextents (MAXEXTNT)	64	1	999	256	Yes	SQLDBGEN
Maximum number of dbspaces (MAXDBSPC)	1000	7	32000	10240	Yes	SQLDBGEN
Catalog dbspace (PUBLIC.SYS0001) Size (4 kilobyte pages)	None	128	8388607	12800	Yes	SQLDBGEN
First package dbspace (PUBLIC.SYS0002) Size (4 kilobyte pages)	None	128	8388607	2048	Yes	SQLDBGEN
HELP text dbspace (PUBLIC.HELPTXT) Size (4 kilobyte pages)	None	2304	8388607	8192	No	SQLDBGEN
ISQL dbspace (PUBLIC.ISQL) Size (4 kilobyte pages)	None	128	8388607	1024	No	SQLDBGEN
SAMPLE dbspace (PUBLIC.SAMPLE) Size (4 kilobyte pages)	None	512	8388607	512	No	SQLDBGEN
Internal dbspaces -Size (each) (4 kilobyte pages) -Number	None None	128 2	8388607 31997	1024 80	No	SQLDBGEN
Initial dbextents -Size (each) -Number	None None	1 cylinder 1	1 volume 999	77 cylinders 1	No	MDISK

Notes:

1. The cylinder specifications listed above for the starter database are for IBM 3380 storage devices. Make the appropriate adjustment for your storage devices.
2. PUBLIC means that the dbspace is publicly owned.
3. Not all dbspaces generated by the starter database are shown in Table 2. For all dbspaces generated for the starter database, see Figure 93 on page 306.

Defining Database Directory Size

The DB2 Server for VM directory (called BDISK) contains control information and page tables for mapping dbspace page references to physical DASD locations. Its size determines the maximum number of dbextent pages and the number of page table entries that can be supported by the database being generated.

If necessary, you can later expand the directory to hold more dbspace pages, or more dbspace and dbextent pages. Refer to “Expanding the Database Directory” on page 159 for more details.

The directory for the database is defined by adding an MDISK control statement to the VM directory entries for a database machine. If Data Spaces Support is used,

4096-byte blocks can be used for the directory, but otherwise the database manager requires the use of 512-byte blocks for its directory. The SQLDBGEN EXEC does the actual formatting of the minidisk. The MDISK parameters you supply determine the number of blocks in the directory minidisk.

Table 3 shows the recommended cylinder (or block) allocations for various DASD device types, based on assumed maximum database sizes.

<i>Table 3. Recommended Directory Allocations for Various Database Sizes</i>					
Directory Space for Various IBM Storage Devices					
Maximum Database Size	3375	3380	3390	9345	FB-512 BLOCKS
10 megabytes	TRK(3)	TRK(3)	TRK(3)	TRK(4)	BLK(124)
50 megabytes	TRK(7)	TRK(6)	TRK(6)	TRK(7)	BLK(310)
100 megabytes	CYL(1)	TRK(11)	TRK(10)	TRK(12)	BLK(496)
500 megabytes	CYL(5)	CYL(4)	CYL(4)	CYL(6)	BLK(2232)
1 gigabyte	CYL(10)	CYL(8)	CYL(7)	CYL(11)	BLK(4480)
2 gigabytes	CYL(19)	CYL(16)	CYL(14)	CYL(21)	BLK(8866)
4 gigabytes	CYL(38)	CYL(32)	CYL(27)	CYL(42)	BLK(17696)
5 gigabytes	CYL(47)	CYL(40)	CYL(34)	CYL(52)	BLK(22080)
10 gigabytes	CYL(92)	CYL(80)	CYL(68)	CYL(101)	BLK(44144)
50 gigabytes	CYL(459)	CYL(400)	CYL(337)	CYL(504)	BLK(220286)

Note: The values in this table apply when the defaults are used for MAXPOOLS, MAXDBSPC, and MAXEXTNT. These parameters are described in “Establishing Database Capacity Parameters” on page 20.

Use Table 3 to choose the initial directory size. Detailed information for generating its values is contained in Appendix B, “Estimating Database Storage” on page 449. When estimating the maximum database size, include the sizes of the public, private, and internal dbspaces.

The directory minidisk for the starter database supports about 4.9 gigabytes of data. This includes space for internal dbspace definitions so the actual space supported for public and private dbspaces is about 4.6 gigabytes.

Directory Allocation Considerations

Maximum Database Size: The directory minimum size cannot extend beyond a single volume; therefore, the maximum database size is limited by the single volume capacity of the device type used. The absolute maximum size for a database is either 64 gigabytes or the limit imposed by the device type, whichever is smaller. For the limits imposed by various devices, see Table 40 on page 451 and Table 41 on page 451.

Placement of Directory: The directory minidisk will be used extensively by the database manager for resolution of data addresses. Thus, you should not allocate it to a volume that will contain either the log minidisks or heavily used data dbextents. Instead, place it on a separate volume to avoid device contention.

If DASD is limited on your system and the directory must share a volume with data dbextents, put it on a volume with a dbextent that contains infrequently referenced data. For example, sharing a volume with private dbspaces or historical data is preferable to sharing one with public dbspaces or current, highly active data.

Defining the Database Log

The database manager requires at least one log minidisk and can support two. It is recommended that you use two logs.

The log minidisks contain information, recorded during database processing, that is used to support database recovery facilities. This includes control information (for example, COMMIT statement and checkpoint records) and the specifics of database changes (for example, inserts, updates, and deletes).

If you define two log minidisk they must be exactly the same size. Do not define them on different device types because it is almost impossible (because of rounding) to get identically sized data sets using space allocation algorithms.

The log history area, which is the final page of the log, is copied to the database machine's A-disk as the file ARIHSDS ARCHIVE immediately after a successful database or log archive. This A-disk file is used during a subsequent restore, if the log history area is unusable due to a log failure. The A-disk file is also copied to the file ARIHSDS PRECLDLG when a COLDLOG RECONFIGURE is done to ensure recoverability.

The size of the log data set is specified by the VM MDISK control statement, as shown in Figure 90 on page 297. The size you specify will depend on the use of the database and on the type of recovery capabilities you want. If you underestimate this size at database generation time, you can redefine it afterwards, as described in "Log Reconfiguration" on page 240 .

Log Size Considerations

The log size depends on the number of changes that you expect will be made to the database and on whether or not you plan to use archiving facilities. If either database or log archiving is enabled, the log must be large enough to hold all the logging done between archives; otherwise it need only be large enough to hold the logging done in a few hours.

Note: If you are putting dbspaces in nonrecoverable storage pools, keep in mind that only minimal logging is done for them, so the following log size considerations would not apply to those dbspaces.

Log Size without Archiving: If you run the database manager without the archiving facilities (LOGMODE=Y or N), log space is reclaimed as applications finish and checkpoints of the database are taken. Usually, this occurs every few seconds or every few minutes. Many uses of the database manager can be supported by a log size of only one or two cylinders; however, a long-running application may require more log space.

Typically, the largest demand for log space is online loading or data reorganization. These processes run longer than most applications and cause a lot of logging to occur.

A starting estimate for the initial log size is twice the space requirements of your largest dbspace. If you have one exceptionally large dbspace, you can disregard it and use the size of the next largest dbspace. The data in the largest dbspace can be loaded and reorganized offline with logging inhibited.

Log Size with Archiving: If you are using the archiving facilities (LOGMODE=A or L), log space is not reclaimed until an archive is taken. That is, log space is not reused between archives of the log or database. Typically, you would only archive the database once or twice a week. You may choose to do log archiving more frequently, depending on database usage.

To estimate the size of the log, consider the amount of logging that will occur between archives. A useful approach is to estimate the percentage of data that will be generated, deleted, and changed over one archive period as follows:

$$\begin{aligned} \text{logsize estimate} = & (\text{percentage generated} \\ & + \text{percentage deleted} \\ & + \text{percentage changed} \times 2) \\ & \times \text{database size} \end{aligned}$$

For example, assume that in a one-week period the database size grows by 5% but also shrinks by 4%, and that 6% of the database (rows) are changed. Your estimate for the log size would be:

$$\text{logsize estimate} = .21 \times \text{database size}$$

If your database size were 100 megabytes and you wanted an archive period of one week, your log size estimate would be:

$$\text{logsize estimate} = 21 \text{ megabytes}$$

This is approximately 30 cylinders of an IBM 3390 DASD device.

Logging Generated by Loading: The log requirements for processing the DBS utility DATALOAD and RELOAD commands in multiple user mode are:

- If the NEW option is used: enough space to hold the log entries for all table rows to be inserted
- If the PURGE option is used: enough space to hold the log entries for all table rows to be deleted as well as for all rows to be inserted.

The log space consumption caused by these operations can be avoided by running the DBS utility in single user mode with LOGMODE=N specified, or by using the COMMITCOUNT option to force periodic checkpoints in multiple user mode.

Placement of Logs: Like the directory minidisk the log minidisks are frequently referenced during processing. To avoid device contention, they should reside on separate volumes from the directory or heavily used dbextents.

Placement of Dual Logs: If two log minidisks are defined, place them on separate volumes. If they were allocated to the same one, loss of that volume would cause the loss of both logs, thus defeating the purpose of dual logging.

Placement of Database A-disk: The database machine A-disk should be on a volume separate from the log minidisks. If it is allocated to the same volume as either log, loss of that volume would result in loss of both copies of the log history area (that is, the one on the log itself and the one on the A-disk) thus defeating the purpose of having two copies of the history area.

Establishing Database Capacity Parameters

The MAXPOOLS, MAXEXTNT, MAXDBSPC, and CUREXTNT keyword control statements can be specified as input to database generation. The SQLDBGEN EXEC calls the program ARISQLDS with STARTUP=C to process these control statements. The first three of these statements are optional. The last one must be specified.

The MAXPOOLS, MAXEXTNT, and MAXDBSPC values are fixed when the database is generated: once defined, they cannot be changed for its lifetime. To avoid future limitation problems, it is recommended that you set them to the allowed maximums. This will take about 1 cylinder of DASD on a 3380 device for the directory, and 280K virtual storage when the database manager is running.

Estimating MAXPOOLS

The MAXPOOLS specification determines the maximum number of storage pools that can be defined in the database. Storage pools control the location of data on DASD volumes - that is, what dbspaces are located on what volumes. You can make a generous estimate for MAXPOOLS, since the value specified results in only a small directory space allocation for each potential storage pool. You should plan on having one storage pool for each user group (or billing account), and one for each major application you expect the database to support.

Estimating MAXEXTNT

The MAXEXTNT controls the maximum number of dbextents that are defined to support the database being generated. Dbextents determine the physical allocation of DASD space for a storage pool.

Because a dbextent is a VM minidisk, it cannot span DASD volumes. This means that you need at least as many dbextents as volumes. You can, of course, define multiple dbextents on one volume. It also means that if you have a dbspace that spans multiple volumes, the corresponding storage pool requires multiple dbextents.

Because you should plan to support multiple dbextents for each storage pool and you should be prepared to extend most, if not all, of your planned storage pools, MAXEXTNT should be much larger than MAXPOOLS. Your estimate for it can be generous because this value results in only a small directory space allocation for each potential dbextent.

Estimating MAXDBSPC

MAXDBSPC controls the maximum number of dbspaces, including internal dbspaces, that can be defined for the database. See "Determining the Internal Dbspace Requirements" on page 23. A dbspace is a logical allocation of database space for holding one or more tables and their indexes. A dbspace is assigned to a storage pool when it is defined and draws on the actual DASD space available in that storage pool on an as-needed basis. Typically, dbspaces are defined to support private space allocations for individual users and space allocations for

specific applications; thus, the number of dbspaces required generally depends on the number of users and the number of tables needed for applications. Each user probably requires from one to five private dbspaces over the lifetime of the database, and each application requires, at most, one dbspace for each table being accessed. For performance reasons, one table per dbspace is recommended.

As with the previous two parameters, your estimate for MAXDBSPC can be generous, because the value you specify will result in only a small allocation of directory space for each potential dbspace.

Estimating CUREXTNT

CUREXTNT determines the number of dbextents defined during database generation. This number should be sufficient to support your initial storage requirements. You can add more dbextents after database generation.

Establishing Initial Dbspace Requirements

Determining the System Dbspace Requirements

Any public dbspace that has SYS as the first three characters in its name is reserved for system use only. The system dbspaces established at database generation time are PUBLIC.SYS0001, PUBLIC.SYS0002, PUBLIC.HELPTXT, PUBLIC.ISQL, and PUBLIC.SAMPLE.

This section presents only general concepts related to setting the initial dbspace sizes. For detailed information, see Appendix B, “Estimating Database Storage” on page 449. For information on how to later increase the size of SYS0001 and SYS0002, see “Specifying Initial Dbspaces” on page 309 .

- PUBLIC.SYS0001 holds the database catalog tables. The size required for it varies considerably, depending on factors such as the number of tables, columns, indexes, views, and users in the database. For guidelines, see “Estimating SYS0001 Dbspace Requirements” on page 456.
Note: Physical space is not actually consumed until required, so you can afford to define the SYS0001 dbspace to be very large. Be generous: this dbspace cannot be dropped or recreated after the database is generated. If you make it too small and SYS0001 runs out of usable space, you will have to regenerate the database which can be a considerable task.
- PUBLIC.SYS0002 holds the definitions of views and packages. This dbspace, which cannot be dropped or recreated after generation, can hold a combination of 255 views and packages. If you anticipate more views and packages than this, you can acquire additional dbspaces after database generation, as described in “Acquiring Dbspaces for Packages” on page 161.
- PUBLIC.HELPTXT holds the online HELP tables. You will need 2304 pages for each IBM-supplied HELP text that you install. The starter database uses 8192 pages.
- PUBLIC.ISQL holds several tables; EXAMPLE.ROUTINE, SQLDBA.ROUTINE, and SQLDBA.STORED QUERIES. An allocation of 1024 pages should be enough for most uses. If you have many users or expect to make extensive use of the ISQL stored queries facility, consider increasing this. See “Estimating ISQL Dbspace Requirements” on page 465.

- PUBLIC.SAMPLE contains copies of the sample tables for ISQL users, to help them gain experience with using the database manager. Usually, every ISQL user has a copy of the sample tables. An allocation of 512 pages should be enough for all your users, but you can increase the size if you have many ISQL users. Alternatively, you can ask experienced ISQL users to drop their copies after they no longer need them to free space for new users' tables.

The ARISDBU MACRO contains SQL statements to acquire the public dbspaces HELPTTEXT, ISQL, and SAMPLE. If you want to increase their size, update the appropriate ACQUIRE DBSPACE statement in ARISDBU.

Except for PUBLIC.SAMPLE, the sizes that you establish for system dbspaces at database generation time can limit the logical capacity of your database. Because physical space is not actually used until required, you should establish large sizes for them. The large recommended sizes shown in Figure 2 will support most uses of the database manager.

System dbspace	Recommended Sizes (in Pages)	Default in pages
SYS0001 (Catalog Tables)	30 + .33 x the number of tables + .40 x the number of views + .10 x the number of columns + .50 x the number of packages + .03 x the number of dbspaces (including package dbspaces) + 10.28 x the number of users + 8.10 x the number of package dbspaces + .25 x the number of character sets + .13 x the number of keys	12,800
SYS000n (packages)	2,048 for each dbspace	2,048
PUBLIC.HELPTTEXT	2.304 x Number of languages installed	8,192
PUBLIC.ISQL	The larger of : 1,024 or (0.88 x the number of stored queries)	1,024
PUBLIC.SAMPLE	512	512

Figure 2. Guidelines for the Sizes of the System Dbspaces

Determining the Initial User Dbspace Requirements

When you generate the database, you need only consider the dbspace requirements for its initial use. To determine the initial user dbspace requirements, either consult with the database administrator or refer to the *DB2 Server for VM Database Administration* manual. The SQLADBSP EXEC can be used to add more later, up to the MAXDBSPC value.

For more information, refer to Chapter 7, "Managing Database Storage" on page 151.

Determining the Internal Dbspace Requirements

The database manager uses internal dbspaces to process commands that require sort operations and to process views that require materialization. For information on sorting and materialization, see the *DB2 Server for VM Database Administration* manual.

The internal dbspaces are held until a COMMIT or ROLLBACK statement is issued; therefore, a single application may hold a number of internal dbspaces at one time. For example, if each SELECT needs an average of two internal dbspaces, and a certain program issues five SELECTs before issuing a COMMIT statement, then that program will hold 10 internal dbspaces. Internal dbspaces that are not in use take up minimal space (approximately 4 bytes of directory space for each page).

Allocate at least 30 internal dbspaces; more if your installation has interactive users. The exact number required depends on the number of logical units of work (LUWs) that are concurrently active and the amount of sorting and view materialization required in those LUWs. Because the number of NCUSERS is comparable to the number of concurrently active LUWs, as a guideline, in addition to the minimum of 30, you may want to provide 10 internal dbspaces for each NCUSER (see the description of the NCUSERS parameter on “NCUSERS” on page 66). After the database has been generated, you can always add more internal dbspaces by using the SQLADBSP EXEC. All internal dbspaces (and their storage pool assignments) are redefined on each run of this EXEC.

The physical placement of the internal dbspaces affects performance, especially when you perform a sort operation on a large table. You should place internal dbspaces in their own storage pool, and use multiple dbextents over multiple devices. There are several ways of doing this. Suppose you had 300 3380-type cylinders for internal dbspace dbextents, you could use one of these strategies:

1. Make the first dbextent small (less than 100 cylinders), and each succeeding dbextent twice the size of the preceding one. For example, have dbextents that are 20, 40, 80, and 160 cylinders in size.
2. Graduate the sizes of the dbextents. For example, have dbextents that are 10, 20, 30, 40, 50, 60, and 90 cylinders in size. The last dbextent is extra large so that unusually large sorts can be accommodated.
3. Have several small dbextents and a few big ones. For example, have five dbextents of 20 cylinders each, and two of 100 cylinders.

The purpose of all these strategies is to spread input/output activity over more devices as the size of a sort increases. The strategy you adopt determines how many dbextents a sort requires. With the first strategy, a sort requiring 60 cylinders uses two dbextents. With the second and third strategies, the same sort requires three dbextents. Use a strategy that is suitable for your organization.

Sorting is done for ORDER BY, GROUP BY, join, CREATE INDEX, or UNION operations. The internal dbspaces must be large enough to hold the rows being sorted. For example, if an ORDER BY operation is requested using all the columns of an entire table, the internal dbspace must be large enough to hold the whole table. Less space is required if all the columns are not selected. During index creation, space is required only for the key columns. To calculate the required size of an internal dbspace, use the formula $(KEYSIZE + 8 \text{ bytes}) * ROWCOUNT$. Make the internal dbspaces large enough to hold the largest table or query result you

want to be able to sort. The dbspace size estimates are discussed under Appendix B, “Estimating Database Storage” on page 449.

The number of internal dbspaces required also depends on the planned usage of the system. Fewer are needed for preplanned application processing than for dynamic query processing, as query users usually hold dbspaces longer than do preplanned applications.

Internal dbspaces can also be stored on a virtual disk. Only use virtual disks for internal dbspaces because information on a virtual disk is lost when the database is restarted. For more information on virtual disk support, see the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

Determining Initial Dbextent Requirements

Sufficient space must be allocated during database generation to support your initial dbspace data storage requirements. You must define at least one dbextent for each storage pool that initially contains dbspaces. The specific amount to allocate for each storage pool depends on the following considerations:

- System dbspace support

System dbspaces are heavily used, so they should not share their storage pool (storage pool 1) with heavily used user dbspaces. Until you gain experience with your data, do not put user dbspaces in the same storage pool as system dbspaces.

You should undercommit storage pool support for the SYS0001 and SYS0002 dbspaces. If the catalog tables grow significantly, you can later allocate an additional dbextent, probably on a separate device, to avoid excessive device contention on catalog access.

Storage pool support for PUBLIC.HELPTTEXT should be large enough to hold the HELP tables; PUBLIC.ISQL must be large enough to hold your initial needs for stored queries; and PUBLIC.SAMPLE should be large enough to hold the number of sample data tables needed.

- End user dbspace support

Dbspaces for use primarily by end users should be supported by one or more storage pools. Public and private dbspaces can share a storage pool; however, you may want to manage space allocation differently for these two cases.

A recommended approach to storage pool support for end user data is to define more dbextent space than is needed to support your initial dbspace definitions. This approach is called overcommitting, and ensures that end user space requirements can be accommodated as existing users need more space or more users are added to the system.

If your installation plans to bill users for DASD storage space, you may want to consider separate storage pools for different user groups (or account numbers).

Note: You can also use statistics from the SYSTEM.SYSDBSPACES catalog table to achieve this.

- Dbspace support for applications

Storage pool support of dbspaces for use primarily by application programs varies, depending on the nature of the data and the storage management technique. In general, consider using different storage pools for different

applications, and undercommitting storage pool support for application dbspaces.

The dbspaces for applications should be defined to be larger than is believed necessary, to avoid later reorganization because of data growth. If you do this, storage pool requirements are smaller than the dbspace sizes indicate. The initial storage pool allocations should be large enough to cover initial loading of the data plus growth over the next planning period (for example, six months or a year).

- Internal dbspace support

Storage pool support for internal dbspaces should be undercommitted, since you probably do not need storage to support all internal dbspaces at the maximum size. As a rough estimate, the storage pool for internal dbspaces should have enough DASD space available to hold data for three internal dbspaces (at the internal dbspace size specified at database generation).

Storage space for internal dbspaces is taken from the storage pool assigned at database generation time. In general, this storage pool should not be used for system dbspaces or other heavily used dbspaces. Consider using a separate storage pool just for internal dbspaces.

For more information on storage organization techniques, see Chapter 7, “Managing Database Storage” on page 151.

Choosing an Application Server Name and VM Resource Identifier

In planning for database generation, you can choose two names for your database. The first name is the server name that the users will specify. The second name is the resid (VM resource identifier) that identifies the application server to VM. The server name and the resid can be the same if the requirements for both are met. If the server name is longer than 8 characters, then you must choose a resid. You must also decide whether the application server can be accessed from other processors.

Note: When using remote access, it is recommended that the system administrator ensure that server names are unique within a set of interconnected SNA networks, and that resids are unique in a TSAF collection or a gateway. (A gateway is also referred to as an LU.) The resid must also be identified with a GLOBAL scope. For more information about these requirements, see “Distributed Processing Administration” on page 147.

The server name must be from 1 to 18 characters. It should start with an alphabetic character which can be followed by alphabetic characters, numeric characters, or underscores. The server name should be unique within a set of networks that are interconnected. The server name is stored in the *resid* SQLDBN file on the production minidisk.

The resid must start with an alphanumeric character and be from 1 to 8 characters. The terms resid and the TPN (transaction program name) are synonymous. The resid is used to identify the database resource to the VM system, and in combination with the NETID and LU name (AVS gateway name) provides the network address of the resource. The resid can also be a 4 byte hex TPN such as the DRDA default TPN x'07F6C4C2'. However, there is little need to define a

hexadecimal resid for an application server. The use of a character resid is preferred because it is more readable.

To specify a value for resid that is different from that specified for the server name, you must create an entry in the RESID NAMES file that is on the accessed production minidisk of the application server. This file correlates the server name and the resid. The resid defaults to the server name if:

- The RESID NAMES file does not exist, or
- The database manager does not find a matching entry in the RESID NAMES file.

For ease of administration, it is best to keep the resid identical to the server name. If the two names are not identical, the VM users accessing the application server must also access a CMS communication Directory that has an entry defined for this server and resid (known as the :dbname and :tpn tags respectively) even if both the user and the application server are in the same TSAF collection.

Choosing the Application Server Default CHARNAME and CCSID

The *application server default CHARNAME* is set using the CHARNAME initialization parameter. The database manager uses the CHARNAME value to determine the classification table and translation table which are used to identify valid characters and to determine how to fold lowercase characters to uppercase. For more information on the CHARNAME initialization parameter, see “CHARNAME” on page 62.

The CHARNAME parameter also specifies the *application server default coded character set identifier (CCSID)*. For a newly installed database, the application server default CHARNAME is INTERNATIONAL, and the application server default CCSID is 500. For a migrated database, the application server default CHARNAME is ENGLISH, and the application server default CCSID is 37. The application server default CCSID is the value of CCSIDMIXED if it is not zero, otherwise it is the value of CCSIDSBCS. Refer to “CCSID Conversion” on page 333 and “Determining CCSID Values” on page 337 for more information on CCSIDs.

If you use DBCS characters, you need to use a mixed CCSID as the the application server default. A mixed CCSID has both an SBCS component CCSID, and a DBCS component CCSID. For more information, see Table 22 on page 337.

The application server default CCSID value is used for the following:

- The CCSID that SQL statements are converted to for processing by the relational data system (RDS) component
- The CCSID of constants (including hexadecimal constants) which are part of the SQL statement processed by the RDS component

Depending on the *application server default subtype* value (that is, the CHARSUB value), the application server default value for CCSIDMIXED or CCSIDSBCS is used for the following:

- The CCSID of special registers which represent character data (for example, CURRENT USER and CURRENT DATE)
- The CCSID of the results of the scalar functions CHAR, DIGITS, and HEX

- The CCSID of the character representation of datetime values (for DRDA protocol, this is always the CCSIDSBCE value)
- The CCSID of character columns created using the CREATE TABLE or ALTER TABLE statements (when the CCSID or subtype clause is not explicitly specified and when package defaults are not specified). See the *DB2 Server for VM Application Programming* manual for more details on package defaults.

It is important that you choose the correct default CHARNAME and CCSID for your installation. The goals of choosing the correct values are to ensure the integrity of character data representation, and to reduce the performance overhead associated with CCSID conversion. The application server and application requester should have the same CCSID value unless there is a specific reason for them to be different.

When the application server and application requester have different CCSID values, character conversion cannot be avoided. This conversion has an associated performance overhead. Performance degradation also occurs if the CCSID conversion causes a sargable predicate to become residual. For example, this can occur on a simple equals predicate like, T1.C1 = T2.C2. For this case, C2 was created prior to migrating to Version 3 Release 3 and has a CCSID of 37. C1 was created using Version 3 Release 4 with the application server default CHARNAME set to INTERNATIONAL (CCSID 500). As a result, since this predicate requires the CCSID conversion of the data in the columns, it is residual. For more information on performance, see the *DB2 Server for VSE & VM Performance Tuning Handbook*.

For example, if your application server is only accessed by local users whose terminal controllers are generated with code page 37 and character set 697 (CP/CS 37/697) for the US ENGLISH characters, then you should set the application server default CHARNAME to ENGLISH. This is because CP/CS 37/697 corresponds to the CCSID of 37 which corresponds to the CHARNAME of ENGLISH.

To eliminate unnecessary CCSID conversion, choose an application server default CCSID to be the same as the CCSID of the application requesters which access your application server most often.

The following is an example of how these two goals can be in conflict.

The situation has these characteristics:

- An application server is accessed by 5 application requesters which are local (that is, they have the protocol parameter set to SQLDS).
- This application server is also accessed by 100 application requesters which are remote (that is, they are using the DRDA protocol).
- The local application requesters have controllers which are defined with CP/CS 37/697 (this corresponds to CCSID 37).
- The remote application requesters use CCSID 285.

If the application server default CHARNAME is set to ENGLISH (CCSID 37), this keeps the data integrity for the local application requesters. However, CCSID conversion overhead is incurred for all remote application requesters who have CHARNAME UK-ENGLISH (CCSID 285).

If the application server default CHARNAME is set to UK-ENGLISH (CCSID 285), this will avoid the CCSID conversion overhead incurred for the remote application requesters, but will cause data integrity problems for the local application requesters. Certain characters will not be displayed correctly for local application requesters. For example, a British pound sign (£) will be displayed as a dollar sign (\$).

These are the trade-offs to consider when choosing your application server default CHARNAME.

For more information on CCSIDs, see the *Character Data Representation Architecture Reference and Registry* manual.

Attention: Immediately following an installation, the application server CHARNAME is set to INTERNATIONAL and the CCSID is 500. Immediately following a migration, the application server CHARNAME is set to ENGLISH and the CCSID is 37. If you do not **choose** your own application server defaults, these settings may not be correct for your system.

For information on how to change the application server default CHARNAME and CCSID, see “Setting the Application Server Default CHARNAME and CCSIDs” on page 338. For information on how to choose the default CCSID for an application requester, see “Setting the Application Requester Default CHARNAME and CCSIDs” on page 341. For a summary of the considerations for changing these values, see “Considerations when changing default CHARNAME and CCSID” on page 320.

Choosing the Application Server Default Character Subtype

The database manager supports three types of character data:

- SBCS
- Mixed
- Bit.

Note: Character refers to data types CHAR, VARCHAR and LONG VARCHAR in this discussion.

Each database has a default character subtype (that is, the CHARSUB value) which can be either SBCS (single-byte character set) or mixed (mixed single and double-byte character set). The default character subtype is the value used for the subtype attribute of any new character column that is created by either the CREATE TABLE statement or the ALTER TABLE statement. The default subtype is used if a subtype is not specified as a package default option or a preprocessing option, and is not specified explicitly using a subtype clause, or implicitly using a CCSID clause.

The CHARSUB value is also used for determining CCSIDs. For more information on CCSIDs, see “Choosing the Application Server Default CHARNAME and CCSID” on page 26, “CCSID Conversion” on page 333, and “Determining CCSID Values” on page 337. For information on how to change the default character subtype, see “Setting the Application Server Default Character Subtype” on page 342.

Choosing the Default CHARNAME and CCSID for Application Requesters

It is important that the appropriate *application requester default CHARNAME* and appropriate *application requester default CCSID* be chosen. The goals of choosing the correct values are to ensure the integrity of character data representation, and to reduce the performance overhead associated with CCSID conversion.

For example, if your terminal controller is generated with code page 37 and character set 697 (CP/CS 37/697) for US ENGLISH characters, then the application requester should set the default CHARNAME to ENGLISH. This is because CP/CS 37/697 corresponds to the CCSID of 37 which corresponds to the CHARNAME of ENGLISH.

The application requester default CCSID is the value of CCSIDMIXED if it is not zero; otherwise, it is the value of CCSIDSBCS. The application requester default CCSID is used for the following:

- The CCSID of SQL statements coded at the application requester
- The CCSID of host variables which represent character data
- The CCSID of character values described by an input or output SQLDA (when the SQLNAME field is *not* used to override the CCSID value)
- The CCSID of character data returned in a DESCRIBE SQLDA
- The CCSID of message tokens returned in an SQLCA

For more information on setting the default CHARNAME for an application requester, see “Setting the Default CHARNAME and CCSIDs for an Application Requester” on page 342. For more information on CCSIDs, see “CCSID Conversion” on page 333 and “Determining CCSID Values” on page 337.

You can avoid the need for all application requesters to specify the default CHARNAME by setting it using the SQLGLOB EXEC. For information on setting the default CHARNAME for **all** application requesters, see “Setting the Default CHARNAME and CCSIDs for All Application Requesters” on page 341.

Preparing for Database Regeneration

If the SYS0001 dbspace ever becomes too small to hold the catalog tables, or if the contents of the directory minidisk or a dbextent minidisk are damaged or destroyed and you do not have archives to restore them, the database can no longer serve your needs and must be regenerated.

The size and complexity of the regeneration task depends on the size and complexity of the database. This task includes:

- Regenerating the database, including any dbspaces, dbextents, and VM minidisks that may have been added since the previous generation
- Using the DBS utility to unload and reload all the data in the database, including the ISQL routines and the ISQL stored queries.
- Reprocessing all application program packages
- Reestablishing the entire authority scheme

- Recreating all views and indexes.

One way to simplify this task is to keep a record of the various types of information you would need to reestablish the operating environment that existed in the previous database. In particular:

- Keep all the ACQUIRE DBSPACE, CREATE TABLE, ALTER TABLE, GRANT, CREATE INDEX, CREATE VIEW, and CREATE SYNONYM statements for the database in EXECs that call the DBS utility. These EXECs can be run easily on the regenerated database.

Note: If these statements are not kept, you can reconstruct them from information available in the system catalog tables. However, this could take a long time for a large production database.

- Keep all of the input control statements for any add dbspace or add dbextent operations. These statements can be used as input to the SQLDBGEN EXEC when it regenerates the database.
- Keep EXECs used to preprocess each application program so that they can be run on the regenerated database (as separate jobs).

Database Generation Worksheet

This section provides a worksheet. Table 4 covers the database generation control statements. Fill it out as you design your database; then refer to it when you define your minidisks or provide control statements to the SQLDBGEN EXEC.

Table 4 (Page 1 of 3). Database Generation Worksheet									
Database Name									
Server Name		_____							
RESID		_____							
Database Scope		_____ (LOCAL or GLOBAL)							
Minidisk Definitions:									
				cylr/	cyls/				
		cuu	devtype	blkr	blks	volser	mode	pr	pw
Directory	MDISK	_____	_____	_____	_____	_____	R	_____	_____
Log Disk 1	MDISK	_____	_____	_____	_____	_____	R	_____	_____
Log Disk 2	MDISK	_____	_____	_____	_____	_____	R	_____	_____
Dbextent 1	MDISK	_____	_____	_____	_____	_____	R	_____	_____
Dbextent 2	MDISK	_____	_____	_____	_____	_____	R	_____	_____
Dbextent 3	MDISK	_____	_____	_____	_____	_____	R	_____	_____
Dbextent 4	MDISK	_____	_____	_____	_____	_____	R	_____	_____

Table 4 (Page 2 of 3). Database Generation Worksheet

Database Capacity Parameters:

CUREXTNT _____ (A value from 1 to 999 must be specified.)
 MAXPOOLS _____ (Default is 32. Value can be from 1 to 999.)
 MAXEXTNT _____ (Default is 64. Value can be from 1 to 999.)
 MAXDBSPC _____ (Default is 1 000. Value can be up to 32 000.)

Nonrecoverable Storage Pools:

POOL _____ NOLOG (Storage pool 1 cannot be specified.)
 POOL _____ NOLOG
 POOL _____ NOLOG
 POOL _____ NOLOG

Database Extent (Dbextent) Placement:

Dbextent Number	Storage Pool (Default is 1)
-----	-----
1	___
2	___
3	___
4	___

Note: The number of dbextents must equal CUREXTNT, but one is required. The MAXEXTNT value determines the maximum number of database extents.

Table 4 (Page 3 of 3). Database Generation Worksheet

Public Dbspaces:

Purpose -----	SIZE (In 4K Pages) -----	Storage Pool (Default is 1) -----
Catalog Tables	_____	1
Packages	_____	_____
HELP Text	_____	_____
ISQL	1024 (minimum)	_____
Sample Tables	512 (minimum)	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

Note: The public dbspaces for the catalog tables, packages, HELP text, ISQL, and the sample tables are required. The catalog tables must be in storage pool 1.

Private Dbspaces:

Purpose -----	SIZE (In 4K Pages) -----	Storage Pool (Default is 1) -----
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

Internal Dbspaces:

Number: _____ Size in 4K Pages: _____ Storage Pool: _____

Note: The MAXDBSPC value determines the maximum total number of public, private, and internal dbspaces possible.

Chapter 3. Planning for Database Migration

If your installation already has a previous release of the database manager installed, you must consider the effect that migration to the new release will have on your existing databases and applications.

You can migrate to a DB2 Server for VM Version 6 Release 1 database from:

- Version 5 Release 1
- Version 3 Release 5
- Version 3 Release 4
- Version 3 Release 3
- Version 3 Release 2
- Version 3 Release 1

Note: If you are on an earlier release, you will have to migrate to Version 3 Release 5 first and then to Version 6 Release 1.

Before migrating:

- Read the discussions on release-to-release incompatibilities in Appendix I, “Incompatibilities Between Releases” on page 503 for changes that may be required in application programs.
- Ensure that the requirements discussed in Chapter 1, “Planning for Installation” on page 1 are met. For information on the actual installation and migration steps, see the *DB2 Server for VM Program Directory*.
- It is strongly recommended that you archive your databases before migrating, so that you can back out of the migration process should it become necessary.
- Consider installing the database manager and generating a starter database to try out the new functions before migrating your existing databases.

This chapter also contains sections on the following:

- Release coexistence considerations

It can be impractical to migrate all the databases in a local or distributed environment to the current level at the same time. For information on the level of coexistence that is possible see “Release Coexistence Considerations” on page 41.

- Migration from a VSE to a VM operating system

It is possible to move a DB2 Server for VSE database to a VM operating system. There is no need to convert the data in a database when you move from VSE to VM; the data is system-independent. You move data from VSE to VM by taking a database archive of the DB2 Server for VSE database and then restoring the database archive tape on the VM operating system.

If you have moved a database from VSE to VM, you may have some VSE application programs that you do not want to convert. These programs can access the databases on VM using VSE guest sharing. For more information, see “Migrating from a VSE to a VM Operating System” on page 42.

- System migration from one IBM VM system to another

You can migrate your databases to a new VM operating system in two ways:

- Archive them in the original VM operating system, install the database manager on your new VM operating system, and restore the databases in the new operating system.
- Install a new VM operating system on the processor you use to access your databases. You can then access your databases as you did in the original operating system.

See the information about the operating systems involved, beginning on page “Migrating from a VM/XA to a VM/ESA Environment” on page 44.

- Converting a service machine to a database machine

For any particular processor, you may need to convert a service machine to a database machine if a database machine is required on that processor.

“Converting a Service Machine to a Database Machine” on page 53 explains this process.

Migration Considerations

For users of an earlier version of the database manager, installing Version 6 Release 1 means loading the new code by running one or more IBM supplied programs, and migrating any existing databases. This section highlights the considerations that you should be aware of when doing this.

The topics are grouped by the release level of the database that is being migrated. Start at your release level and read to the end of this chapter. For example, if your database is Version 3 Release 1, you must review all the topics; if it is Version 3 Release 2, you need only read from that topic to the end of the chapter.

Increasing the HELPTEXT Dbspace

A database that is migrated keeps its existing HELPTEXT dbspace, which may not be large enough to support the Version 6 Release 1 HELP text. The size required for this dbspace depends on the number of national languages for which you have HELP text. It should be:

2,304 pages x number of languages installed.

This dbspace can be increased at any time before you install the current HELP text. For information, see the *DB2 Server for VM Database Administration* manual.

If users are running their applications under the DRDA protocol, some database manager facilities are not supported. For a list of these restrictions, see Appendix H, “DRDA Considerations” on page 499.

Migrating from Version 3 Release 1

Considerations for Invalid Indexes

Before you migrate, at least four dbspace blocks must be available in the database directory to allow for expansion of the invalid entities table. During migration, any entries in the invalid entities table are migrated to the new format. The new table format requires additional space in the directory. If there are any entries in the invalid entities table, it is possible that there may not be enough room in the directory to allow the table to be modified during migration.

For information about directory space verification, see the *DB2 Server for VM Program Directory* .

Conversion of Packages

After migration, all packages are dynamically reprocessed on first use. This conversion can cause a performance degradation over the first few days as the packages are referenced and reprocessed.

To help minimize this degradation, the REBIND PACKAGE command is provided so that all packages can be recreated, if desired, after migration but before production. For information about this command, see the *DB2 Server for VSE & VM Database Services Utility* manual.

You can also convert a package by explicitly reprocessing the application program. Before reprocessing your applications, you should be aware of any statements that may behave differently with the new release. See 'Release to Release Incompatibilities' in Appendix I, "Incompatibilities Between Releases" on page 503.

Migrating from Version 3 Release 2

When migrating from a Version 3 Release 2 database, you may want to update the SNA NETID file. For information on this task, see "Updating the SNA NETID File" on page 13.

With Version 3 Release 3, you can specify a server name of up to 18 characters, and a resid that is correlated with it. For more information on the conventions to follow when specifying the server name and resid, see "Choosing an Application Server Name and VM Resource Identifier" on page 25.

Choosing an Application Server Default CHARNAME

After migration, the database manager sets the application server default CHARNAME to ENGLISH, and sets the application server CCSID values as follows:

- CCSIDSBBCS = 37
- CCSIDMIXED = 0
- CCSIDGRAPHIC = 0.

You can change the value of the default CHARNAME, which in turn determines the values for the three application server default CCSIDs. These four values are stored in the VALUE column of the SYSTEM.SYSOPTIONS catalog table. The corresponding values in the SQLOPTION column for these defaults are CHARNAME, CCSIDSBBCS, CCSIDMIXED, and CCSIDGRAPHIC.

The value you choose for the default CHARNAME should accurately reflect the type of data that will be stored in the database: that is, the type of code page and character set that describes the data, and whether or not the database manager is to support DBCS characters or MBCS characters, or both. For more information, see "Character Set Considerations at Startup" on page 62, "Determining CCSID Values" on page 337, and "CCSID Conversion" on page 333. For a summary of the considerations for changing these values, see "Considerations when changing default CHARNAME and CCSID" on page 320.

Setting Migration CCSID Values

After choosing your default CHARNAME, you must also set your CCSID values for character and graphic data that existed before the migration to Version 3 Release 3. The CCSID value of character and graphic data stored in tables that were created before Version 3 Release 3 are specified by the three other rows (with SQLOPTION value MCCSIDSBCS, MCCSIDMIXED and MCCSIDGRAPHIC) in the SYSTEM.SYSOPTIONS catalog table. The migration CCSID values (MCCSIDSBCS, MCCSIDMIXED, and MCCSIDGRAPHIC) are used for single byte, mixed, and graphic data that was created prior to Version 3 Release 4 and therefore does not have a CCSID associated with it. The database manager sets the migration CCSID values as follows:

- MCCSIDSBCS = 37
- MCCSIDMIXED = 0
- MCCSIDGRAPHIC = 0.

If the code page and character set used to create the migrated data (that is, the data that was inserted into the database prior to Version 3 Release 3) is not CP/CS 37/697, these settings are not correct for your installation and **must** be changed. You can determine the CCSIDs for migrated data from the code page and character set that was used to generate the terminal controller where the data was entered.

For an example of how your choice of migration CCSID value affects the characters displayed, refer to page 339.

To determine if your database contains graphic or mixed data, issue the following query:

```
SELECT COUNT(*) FROM SYSTEM.SYSCOLUMNS
WHERE COLTYPE = 'GRAPHIC' OR
      COLTYPE = 'VARGRAPH' OR
      COLTYPE = 'LONGVARG' OR
      SUBTYPE = 'M'
```

If the query returns a result of zero rows, the database contains neither graphic nor mixed data; a nonzero result indicates the number of columns in your database that do contain such data.

Handling SBCS Data: If your database contains only SBCS data (that is, the above query returns a result of zero) prior to Version 3 Release 3, the migrated CCSID values for mixed and graphic data (MCCSIDMIXED and MCCSIDGRAPHIC) must remain 0.

If the MCCSIDSBCS value of 37 is not correct for your installation, this must be changed to correspond to the code page and character set used to create the migrated data. For example, if the data was created with CP/CS 273/697 (GERMAN), the CCSID value you should use is 273. For a list of some of the SBCS CCSIDs and their character set and code page values, see Table 22 on page 337.

The row that you must update for data in tables created before Version 3 Release 3 is:

- SQLOPTION='MCCSIDSBCS'

Change the value in the VALUE column to the appropriate SBCS CCSID (for example, 273 for GERMAN). The following statements show how to update or insert the row using this value:

```
UPDATE SYSTEM.SYSOPTIONS SET VALUE = '273'  
WHERE SQLOPTION = 'MCCSIDSBCS'
```

```
INSERT INTO SYSTEM.SYSOPTIONS VALUES  
( 'MCCSIDSBCS', '273',  
  'DEFAULT CCSID FOR MIGRATED SBCS CHARACTER COLUMNS' )
```

Handling Mixed Data: If your database contains graphic or mixed data prior to Version 3 Release 3, you must update the VALUE column of SYSTEM.SYSOPTIONS for the row where SQLOPTION='MCCSIDMIXED' with the appropriate nonzero CCSID value. You **must** also update the row where SQLOPTION='MCCSIDSBCS' to the value of the SBCS component of the mixed CCSID, and the row where SQLOPTION='MCCSIDGRAPHIC' to the value of the DBCS component of the mixed CCSID. If these CCSIDs do not correspond to the components of the mixed CCSID, the wrong conversion selection tables are being used. For a list of some of the mixed CCSIDs and their component SBCS and DBCS CCSIDs, see Table 22 on page 337.

The rows that you must update for data in tables created before Version 3 Release 3 are:

- SQLOPTION='MCCSIDMIXED'

Change the value in the VALUE column to the appropriate mixed CCSID. If you used DBCS characters before Version 3 Release 3, specify the appropriate CCSID value. For example, if you used Kanji characters, specify the value 5035. The following statements show how to update or insert the row using this value:

```
UPDATE SYSTEM.SYSOPTIONS SET VALUE = '5035'  
WHERE SQLOPTION = 'MCCSIDMIXED'
```

```
INSERT INTO SYSTEM.SYSOPTIONS VALUES  
( 'MCCSIDMIXED', '5035',  
  'DEFAULT CCSID FOR MIGRATED MIXED CHARACTER COLUMNS' )
```

- SQLOPTION='MCCSIDSBCS'

Change the value in the VALUE column to the appropriate SBCS CCSID. If you used DBCS characters before Version 3 Release 3, you must specify the SBCS component CCSID of the MCCSIDMIXED value. For example, if MCCSIDMIXED is set to 5035, specify 1027. The following statements show how to update or insert the row using this value:

```
UPDATE SYSTEM.SYSOPTIONS SET VALUE = '1027'  
WHERE SQLOPTION = 'MCCSIDSBCS'
```

```
INSERT INTO SYSTEM.SYSOPTIONS VALUES  
( 'MCCSIDSBCS', '1027',  
  'DEFAULT CCSID FOR MIGRATED SBCS CHARACTER COLUMNS' )
```

- SQLOPTION='MCCSIDGRAPHIC'

Change the value in the VALUE column to the appropriate graphic CCSID. If you used DBCS characters before Version 3 Release 3, this value must be the DBCS component CCSID of the MCCSIDMIXED value that you used. For example, if you used Kanji characters, specify 4396. The following statements show how to update or insert the row using this value:

```
UPDATE SYSTEM.SYSOPTIONS SET VALUE = '4396'  
WHERE SQLOPTION = 'MCCSIDGRAPHIC'
```

```
INSERT INTO SYSTEM.SYSOPTIONS VALUES  
( 'MCCSIDGRAPHIC', '4396',  
  'DEFAULT CCSID FOR MIGRATED GRAPHIC COLUMNS' )
```

Choosing the Default CHARNAME for All Application Requesters

After migration, the application requester default CHARNAME is set to ENGLISH, and the application requester CCSID values are as follows:

- CCSIDSBCS = 37
- CCSIDMIXED = 0
- CCSIDGRAPHIC = 0.

To ensure the integrity of character data representation and to reduce the performance overhead associated with CCSID conversion, it is important to choose the appropriate CHARNAME for the code page used by each application requester. If you need to, you can later change it for *all* application requesters by using the global default SQLGLOB EXEC. See “Choosing the Default CHARNAME and CCSID for Application Requesters” on page 29 and “Setting the Application Requester Default CHARNAME and CCSIDs” on page 341. For more general information on CCSIDs, see “CCSID Conversion” on page 333 and “Determining CCSID Values” on page 337.

Considerations for Mixed Primary Keys with Field Procedures

If you are migrating from Version 3 Release 1 or Version 3 Release 2, the value of CCSID in SYSTEM.SYSKEYCOLS is NULL. For some primary keys, this value is not correct. In this case, you should drop and recreate the primary keys, which you can identify by running the ARISFPKY EXEC after migrating. (For information on this procedure, see the *DB2 Server for VM Program Directory*.)

Migrating from Version 3 Release 3

Considerations for EXPLAIN Tables

Several changes and enhancements were made to the EXPLAIN tables in Version 3 Release 4. If you have existing EXPLAIN tables they must either be renamed, or, dropped and recreated before using the EXPLAIN statement.

An IBM-supplied macro, ARISEXP, recreates the EXPLAIN tables for you.

For additional information on using EXPLAIN tables, see the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

Considerations for VSE Guest Sharing

VSE batch applications can access an application server on VM that is either remote or local. If the application server is in a remote network, the SET XPCC TARGET SYSARI command in the VSE IPL procedure must be replaced by the SET APPCVM TARGET command. If the application server is local, the SET XPCC TARGET SYSARI command in the VSE IPL procedure is not needed, and can be deleted.

Regardless of whether the application server is remote or local, an entry in the DBNAME directory may also be necessary to map the DBNAME to the resid when the DBNAME is greater than 8 characters, or when the DBNAME and the resid are different. For more information on the DBNAME directory, see the *DB2 Server for VM System Administration* manual.

Considerations for the VM Data Spaces Support (VMDSS)

If VMDSS was installed prior to migration, the VMDSS code must be link-edited with the DB2 Server for VM code before running the migration utilities. See the *DB2 Data Spaces Support* manual for more information.

Migrating from Version 3 Release 4

Considerations for Assembler Even Precision Packed Decimal

Prior to Version 3 Release 5, assembler host variables declared as even precision packed decimal were converted to odd precision by the preprocessor. As of Version 3 Release 5, the database manager supports assembler host variables defined as even precision packed decimal, and they are not converted to odd precision. In some cases, the lack of conversion may cause a datatype mismatch between a host variable and a column. To prevent potential performance degradation, applications affected by this change should be modified so the

datatypes of the host variables exactly match the datatypes of the columns to which they will be compared.

Considerations for SQLSTATE Changes for SQL92 Support

The SQLSTATEs returned by several conditions were changed to comply with SQL92. Application programs that have a dependency on the SQLSTATE returned may be affected by these changes. See *DB2 Server for VM Messages and Codes* for information on the changed SQLSTATEs.

Migrating from Version 3 Release 5

Considerations for Uncommitted Read

Prior to Version 5 Release 1, the database manager accepted isolation level uncommitted read as a preprocessor parameter, but internally the isolation level was escalated. As of Version 5 Release 1, isolation level uncommitted read is fully supported. However, this isolation level to take effect, packages that were prepped with uncommitted read in a previous release must be explicitly reprocessed after migration.

Considerations for VMSES/E

As of Version 5 Release 1, installation and service of the database manager code is done using VMSES/E, which is a component of the VM operating system. Several administrative and maintenance processes have changed as a result, including:

- Defining and loading saved segments
- Adding a primary database machine
- Moving an application server to another VM ID
- Installing your own date, time, and accounting exits
- Enabling the DRDA code
- Enabling the DSS code
- Installing NLS support.

Considerations for Support of ESA-mode Processors Only

Any user exits (date, time, or accounting), field procedures, or applications that run in single user mode that are dependent on running in a 370 mode virtual machine must be converted to execute in an ESA mode virtual machine. AMODE 24 is still supported, however it cannot be used if the database is started with SYNCNT=Y. If the database is started with SYNCNT=Y (which is possible only in multiple user mode), exits and field procedures must run with AMODE 31. If the database is started with SYNCNT=N, exits, field procedures, and single user mode applications that require AMODE 24 can be used.

Considerations for the Renaming of the Product

The text of several messages was modified as the result of the renaming of the product. Applications with dependencies on the text of messages may be affected.

Considerations for the Removal of the User Facility Subset

The User Facility Subset is no longer supported; machines on which the subset was previously installed must now contain the full product.

Migrating from Version 5 Release 1

Considerations for RDS Above 16M

In DB2 Server for VM Version 6 Release 1, the RDS component is loaded above 16M whenever possible. As a result,

- If you use AMODE 24, you must use a maximum virtual storage size of 16MB.
- If you use saved segments, it is highly recommended that the RDS and DBSS segments not be put in the same segment space, as this would force the RDS segment to be located below 16M. See “Using Saved Segments for Components” on page 181.

Considerations for TCP/IP

In DB2 Server for VM Version 6 Release 1, the database manager will attempt to use TCP/IP for communications by default. If you do not want to use TCP/IP, specify the initialization parameter TCPSPORT=0.

Release Coexistence Considerations

For installations with multiple databases, you should migrate all your databases to the current level. All users have the same features available to them, and future database migrations are easier.

Applications at the Version 3 Release 3 or higher level can access application servers at the Version 2 Release 2 level, and all subsequent releases. These applications cannot use functions unavailable at the release level of the application server accessed.

Application programs can access databases at higher release levels. These applications can use only functions available at the release level of the preprocessor used.

All existing applications that accessed a database before the database was migrated to another release level continue to work after migration.

Version Compatibility Matrix

Table 5 on page 42 shows the functions that are supported when different releases of the database manager are installed.

Application Requester	Coexistence Considerations	Application Server	
		V2	V3 or V5
V2	Preprocessing	Yes	Error
V2	Processing load module created in V2	Yes	Yes
V2	Processing load module created in V3	Yes	Yes
V2	Guest Sharing	Yes	Yes
V2	Reload package created in V2	Yes	Yes
V2	Reload package created in V3	Error	Yes
V2	Unload/Reload table	Yes	Error
V2	DATALOAD table	Yes	Error
V2	DATAUNLOAD table	Yes	Yes
V3, V5 or V6	Preprocessing	Error	Yes
V3, V5 or V6	Processing load module created in V3	Yes	Yes
V3, V5 or V6	Guest Sharing	Yes	Yes
V3, V5 or V6	Reload package created in V2	Yes	Yes
V3, V5 or V6	Reload package created in V3	Error	Yes
V3, V5 or V6	Processing load module created in V2	Yes	Yes
V3, V5 or V6	Reload table	Yes	Yes

Notes:

1. You cannot use any new release facilities from ISQL, DBS utility or application programs when the application server is running a different level of DB2 Server for VSE & VM than the application requester.
2. See Appendix I, "Incompatibilities Between Releases" on page 503 for incompatibilities that exist between each release and the next release.
3. See Appendix H, "DRDA Considerations" on page 499 for restrictions that apply with DRDA protocol.

Migrating from a VSE to a VM Operating System

This section describes two operating environments on VSE from which you can migrate a database:

- Stand-alone VSE system. VSE is the only operating system on the processor. The database manager runs under VSE.
- VSE runs as a guest operating system under VM. The database manager runs entirely under VSE.

When you move the database from VSE, you can move it to one of these operating systems:

- VM operating system with VSE guest sharing. In this situation, VSE is a guest operating system under a VM operating system, and the database machine is on VM.
- VM/ESA ESA Feature operating system. The database machine runs under a VM/ESA operating system with the ESA Feature.

If you do not have a database on VM, you can migrate your database from a VSE to a VM operating system by archiving the database on VSE, generating a new database on VM, and then restoring the archived database on VM. For more information, see “Moving a Database” on page 49.

If you already have a database on VM, you can move the data in your VSE database to the VM database using the UNLOAD and RELOAD commands. For more information on the UNLOAD and RELOAD commands, see the *DB2 Server for VSE & VM Database Services Utility* manual. If you move the data this way, you have to unload and reload all dbspaces and packages. When the database is on VM, you must recreate all views and indexes, and reestablish the authorities and privileges each user has with GRANT and REVOKE commands for the tables moved from VSE. This is similar to regenerating a database. For more information, see “Preparing for Database Regeneration” on page 29.

When migrating from a prior release, you may want to update the SNA NETID file. For information on this task, see “Updating the SNA NETID File” on page 13.

Moving a Database from a VSE to a VM Operating System

Before attempting to move a database from a VSE to a VM operating system, you should understand the database manager on VM as well as on VSE. You must know how to define and generate a database on VM before you can move a VSE database to VM.

Choosing a VM Resource Identifier

In a DB2 Server for VM database, you can choose a VM resource identifier (resid) in addition to the server name. The resid identifies the application server to VM.

For more information on this, see “Choosing an Application Server Name and VM Resource Identifier” on page 25.

Converting Data in the Database

There is no need to convert the data in a database when you move the database from a VSE to a VM operating system; the data is system-independent. You must have the same release level of the database manager installed on both operating systems, otherwise the database manager at the lower release level must be migrated first. Once both databases are using the same release level, archive the database on VSE and restore it on VM.

Converting Packages

Although the database manager itself does not require you to convert user-created packages when you move from VSE to VM, any program that is moved will have to be compiled and linked again in the VM environment. If you have revised any programs, you should reprocess these programs. Remember to use the preprocessor KEEP option to retain existing authorizations on the program.

Views are stored as packages. These packages do not need to be recreated.

Converting Programs

The VSE programs moved to the VM operating system must be recompiled and linked in the VM operating system. Some of the programs can be run in a CMS/DOS environment without modification. The CMS/DOS environment does not support all VSE macros and functions; some programs must be recoded. Programs that do not need to be recoded still have to be preprocessed, compiled, and link-edited in the VM environment.

The CICS/VSE programs cannot be converted to the CMS environment; they must be rewritten to be used in CMS.

VSE Databases Coexisting under VM

You might choose to move some VSE databases to VM. In such cases, you can run the database on VM, with VSE running as a guest under VM. Users and applications in VSE can also access databases on VM through VSE guest sharing. For more information on guest sharing, see “VSE Guest Sharing Configuration” on page 314.

If you have databases on VSE, use the DB2 Server for VSE manuals. See the *DB2 Server for VM Master Index and Glossary* manual for a list of these manuals.

Migrating from a VM/XA to a VM/ESA Environment

You can migrate your VM/XA databases to a VM/ESA operating system in two ways:

- Archive the databases in the VM/XA operating system, and restore them in the VM/ESA operating system. See “Moving a Database” on page 49.
- Install a VM/ESA operating system on the processor that you use to access your databases. See “Installing Another IBM VM System on Your Processor” on page 49.

You may have to update the SNA NETID file. See “Updating the SNA NETID File” on page 13.

Delaying the Directory and Database Name Changes

If you have just installed a VM/ESA operating system, and you want to delay the directory and database naming changes, you can operate your database machines in non-APPC/VM mode. In this mode, IUCV is used and the remote unit of work and database switching capabilities are not available.

To indicate that the database machine is to use IUCV communication paths instead of APPC/VM, set the DBMODE initialization parameter to N when you start the application server. This connects you to an application server using the database machine's VM ID. Otherwise, APPC/VM paths will be used, and you will be connected to the application server using the resource identifier (*resid*).

If you have already made the directory changes and database name changes, do not set DBMODE to N.

Setting up the Database Machine Directory Entry

To use Version 6 Release 1, ensure that you have the necessary VM directory control statements. An example of these statements is shown in Figure 3.

Note: Only those statements that differ from the ones used in the VM/XA operating system are explained following the figure. For details of the other statements, see “Adding a Primary Database Machine” on page 287.

```
1 ---> USER SQLMACH sqlmachpw xM xM G
      ACCOUNT nnnnnnnn
2 ---> OPTION MAXCONN 26
      IUCV ALLOW
3 ---> IUCV *IDENT SQLDBA GLOBAL
      IPL CMS PARM AUTOCR
      CONSOLE 009 3215 T OPERATOR
      SPOOL 00C 2540 *
      SPOOL 00D 2540 A
      SPOOL 00E 1403
      LINK MAINT 190 190 RR
      LINK MAINT 19D 19D RR
      MDISK 191 3380 cylr 010 volser W
      MDISK 193 3380 cylr 060 volser R rsq1 wsq1
      MDISK 195 3380 cylr 020 volser RR rsq1 wsq1 msq1
      MDISK 200 3380 cylr 034 volser R rsq1 wsq1
      MDISK 201 3380 cylr 008 volser R rsq1 wsq1
      MDISK 202 3380 cylr 077 volser R rsq1 wsq1
```

Figure 3. Example VM Directory Control Statements for the SQLMACH Machine

Statement 1: USER SQLMACH *sqlmachpw* xM xM G

This statement defines the database machine SQLMACH with the VM privilege class G. The recommended minimum size required for SQLMACH is 8MB for the base code only, or for the base code and the DSS code. If the DRDA code is installed, the recommended minimum size is 10MB.

Statement 2: OPTION MAXCONN 26

The MAXCONN value must be increased by 1 (over that specified for the VM/XA operating system), because the machine now makes one additional IUCV connection to *IDENT.

The default value for MAXCONN is 4 in the VM/XA operating system, and 16 in the VM/ESA operating system.

Statement 3: IUCV *IDENT SQLDBA GLOBAL

In a VM/ESA operating system, the database machine is the resource owner, so it must be authorized to connect to the VM system service *IDENT. This authorization is granted by the IUCV entry in the database machine directory. The name of the database (specified in the DBNAME parameter of the SQLDBINS EXEC) is used as the resource identifier.

Figure 4 on page 46 shows the syntax of the IUCV *IDENT statement.

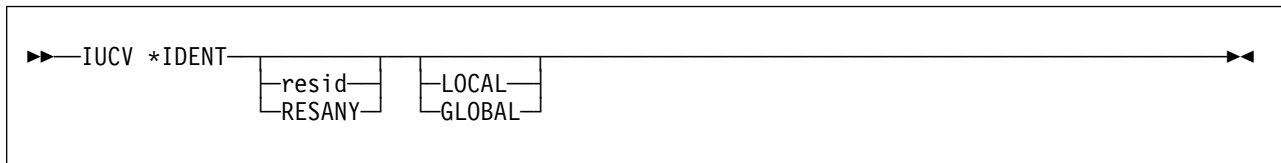


Figure 4. IUCV *IDENT Syntax

resid

This variable is the resource identifier of an application server that can be started in this virtual machine. The machine can have multiple resid entries in its directory. (In Figure 3 on page 45, the resid is SQLDBA.) Usually, the resid is the server name of the application server. However, if the resid and the server name are different, they must both be defined in the RESID NAMES file during database generation.

RESANY

This parameter enables the database machine to identify any resource identifier as either a LOCAL or GLOBAL resource. Specify it if you want to access more than one application server (accessed one at a time).

LOCAL

This parameter ensures that only the application requesters that are on the same processor as the database machine can use this application server

GLOBAL

This parameter identifies an application server as a resource that can be accessed by all application requesters in a network.

Example of a Database Machine Directory with Multiple Databases

Figure 5 shows the control statements in the directory of a database machine with multiple databases. This database machine can manage three application servers on this processor, but only one at any given time. The three database resids in this example are SQLRES1, SQLRES2, and SQLRES3. The first two can only be accessed by users on the local processor, while the third can be accessed by both local and remote users.


```

---> USER SQLMACH sqlmachpw xM xM G
      ACCOUNT nnnnnnnn
      OPTION MAXCONN 26
      IUCV ALLOW
---> IUCV *IDENT SQLRES1 LOCAL
---> IUCV *IDENT SQLRES2 LOCAL
---> IUCV *IDENT SQLRES3 GLOBAL
      IPL CMS PARM AUTOOCR
      CONSOLE 009 3215 T OPERATOR
      SPOOL 00C 2540 *
      SPOOL 00D 2540 A
      SPOOL 00E 1403
      LINK MAINT 190 190 RR
      LINK MAINT 19D 19D RR
      MDISK 191 3380 cylr 010 volser W
      MDISK 193 3380 cylr 060 volser R rsq1 wsq1
      MDISK 195 3380 cylr 020 volser RR rsq1 wsq1 msq1
      MDISK 200 3380 cylr 034 volser R rsq1 wsq1
      MDISK 201 3380 cylr 008 volser R rsq1 wsq1
      MDISK 202 3380 cylr 077 volser R rsq1 wsq1
      MDISK 203 3380 cylr 034 volser R rsq1 wsq1
      MDISK 204 3380 cylr 008 volser R rsq1 wsq1
      MDISK 205 3380 cylr 077 volser R rsq1 wsq1
      MDISK 206 3380 cylr 034 volser R rsq1 wsq1
      MDISK 207 3380 cylr 008 volser R rsq1 wsq1
      MDISK 208 3380 cylr 077 volser R rsq1 wsq1

```

Figure 5. Database Machine Directory Entries

Setting Up the User Machine Directory Entry

In a VM/ESA operating system, the database manager uses advanced-program-to-program-communications/virtual machine (APPC/VM) in place of IUCV. User machines connect to a resource, not to the database machine. A change is required if access to the resource had been controlled by specifying IUCV *dbmachid* in the directory entries of the user machines. The IUCV *dbmachid* must be replaced with the IUCV *resid* statement in each virtual machine directory, to allow the user machine to connect to the application server identified as a resource. Figure 6 on page 48 shows an example of the VM directory entry for a user machine.

Note: Only those statements that differ from the ones used in the VM/XA operating system are explained following the figure. For information on user machine directory entries, see “Defining Additional User Machines” on page 294. For a complete description of VM directory control statements, refer to the *VM/ESA: Planning and Administration* manual.

```

1 ---> USER SQLUSER sqluser xM xM G
        ACCOUNT nnnnnnnn
2 ---> IUCV resid
        IPL CMS PARM AUTOOCR
        CONSOLE 009 3215
        SPOOL 00C 2540 *
        SPOOL 00D 2540 A
        SPOOL 00E 1403
        LINK MAINT 190 190 RR
        LINK MAINT 19D 19D RR
        MDISK 191 3380 cylr 003 volser W
        LINK SQLMACH 195 195 RR

```

Figure 6. Example VM Directory Entries for a User Machine

Statement 1: USER SQLMACH *sqlmachpw* xM xM G

This statement defines the user machine with the VM privilege class G. The user machine requires 6 megabytes.

Statement 2: IUCV *resid* (used for the VM/ESA operating system)

This statement is only required if the IUCV ALLOW control statement is *not* present in the VM directory for the database machine (SQLMACH). Since the default arrangement is for IUCV ALLOW to be specified in the VM directory entry for the database, most users omit this statement. If you later decide to have more control over user machine-to-application server communications, you can change the IUCV control statements.

Figure 7 shows the syntax of the IUCV statement used for the VM/ESA operating system.

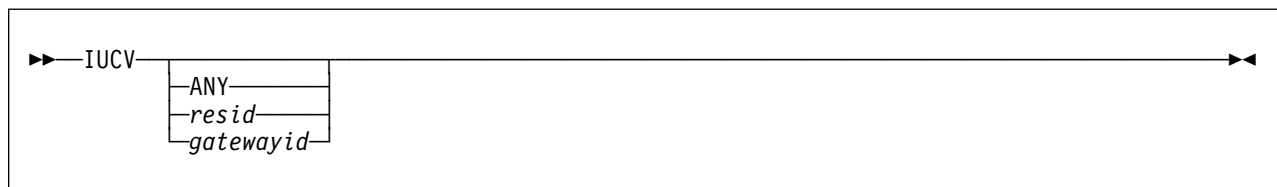


Figure 7. IUCV Statement Syntax

ANY

This parameter authorizes the user machine to connect to any application server identified as a resource.

resid

This variable authorizes a user machine to connect only to the application server identified by *resid*. If more than one IUCV *resid* statement is specified in the machine's directory, the user machine can communicate with more than one application server.

gatewayid

This variable authorizes the user machine to connect to the resources in an SNA network through *gatewayid*, rather than to a specified virtual machine.

For more information about the VM directory control statements that affect intermachine communications, see "VM Directory Control Statements" on page 146.

Database Naming Considerations

You may have to change the names of your databases (*server-name*), to ensure that they are unique within a set of interconnected SNA networks, and that their resid is unique in a TSAF collection or gateway. For more information, see “Distributed Processing Security” on page 147.

When you migrate a database from a VM/XA to a VM/ESA operating system you can specify a value for *server-name* of up to 18 characters, and a value for *resid* of up to 8 characters. For more information, see “Choosing an Application Server Name and VM Resource Identifier” on page 25.

Migrating from a VM/SP to a VM/ESA Operating System

You can migrate your VM/SP databases to a VM/ESA operating system in two ways:

- Install the VM/ESA operating system on the processor you use to access your databases. See “Installing Another IBM VM System on Your Processor.”
- Archive the databases in VM/SP, and restore them in the VM/ESA operating system. See “Moving a Database.”

If you have user exits or single user mode applications that do not support 31-bit addressing, these applications must be converted before the database manager can run AMODE(31).

Installing Another IBM VM System on Your Processor

You can access your databases in another IBM VM system by installing that VM operating system on the processor on which the databases are located. Before doing so, you should archive your databases.

For information on installing VM/ESA, see the *VM/ESA: Installation Guide* manual.

Moving a Database

To move a database to another database manager:

1. Start the source application server.

If you normally use the database manager archiving facility, specify the LOGMODE=A initialization parameter to archive the database, or LOGMODE=L to archive the log. If you do not use the archiving facility, specify LOGMODE=Y.

2. Set the password for authorization ID SQLDBA in the source application server to SQLDBAPW.
3. Create a database archive tape file by issuing the SQLEND ARCHIVE DVERIFY operator command. If LOGMODE is set to L, the database manager also takes a log archive. You cannot use a database archive created by user facilities when moving your database.

Do not destroy the source database until you are certain that it has been correctly moved to the target.

4. Install the database manager that you are going to use as the target (if you have not already done so).

Before proceeding to move the database, it is recommended that you first install and verify the IBM-supplied starter database on the target database manager, to ensure that the target database manager has been correctly installed. For information on how to do this, see the *DB2 Server for VM Program Directory*.

5. Define and generate a database on the target system. The new database directory and dbextents must be defined with sufficient space to contain the old directory and dbextents.

If you are moving from a VSE to a VM operating system, you must **increase** the space allocations used by approximately 16% for count-key-data DASDs that are 10 cylinders or fewer, and for FBA devices that are 5000 blocks or fewer. For data sets larger than 10 cylinders or larger than 5000 FBA blocks, increase the allocation by about 3%. These increases account for VM DASD block I/O. If you use allocations on VM that are the same size as those you used on VSE, the VM database will be too small.

You must define exactly the same number of dbextents and logs that existed on the old database.

For database planning information, see Chapter 2, "Planning for Database Generation" on page 15. For the database definition and generation procedure, see the *DB2 Server for VM Program Directory DB2 Server for VSE Installation*.

6. Perform coldlog processing against the target database manager, by entering the command:

```
SQLLOG DBNAME(server-name)
```

Respond CONTINUE to message ARI0688D (for single logging) or ARI6129D (for dual logging). Respond 0 to message ARI0944D to reformat the log.

7. Restore the database archive tape file created in step 3 above to the target database, by entering the command:

```
SQLSTART DBNAME(server-name) PARM(STARTUP=R)
```

Do not specify LOGMODE=A or L when you issue SQLSTART. Message ARI0253D is displayed, indicating that the restored database archive is not known to the database manager. (The database manager keeps track of archives in the log history area.) Reply IGNORE to this message. When the application server is started and ready for operator commands, shut it down by issuing the SQLEND command.

8. Install the correct version of the HELP text into the target database.

This text is different on the VSE and VM operating systems, so if you have moved from one system to the other and had the HELP text installed, replace it with the target system version.

For information on how to install the HELP text, see the *DB2 Server for VM Program Directory*.

Moving a VM Application Server from One User ID to Another

If you are moving a database from one VM user ID to another VM user ID, you need to do the following:

1. Update the VM directory entries for the new user ID. Use the same statements as they were used in the origin user ID. See “VM Directory Control Statements” on page 146 for some example entries of the VM directory control statements.

VMSES/E Consideration

If the database you are moving is the original database machine (SQLMACH), which VMSES/E recognizes, you must update the installation user ID's VM directory to link to the new production and service minidisks.

End of VMSES/E Consideration

2. Copy all the files from the original service and production minidisks to the new service and production minidisks.
3. Copy the PROFILE EXEC from the original database user ID's A-disk to the new user ID's A-disk.
4. In the new user ID, access the production minidisk as file mode Q in write mode.
5. XEDIT the file ARISPIDC MACRO Q. Change the original database user ID to the new user ID. For example:

Before:

```
Line 1: PRODUCTION:  SQLMACH  195
Line 2: SERVICE:     SQLMACH  193
      :
```

After:

```
Line 1: PRODUCTION:  NEWMACH  195
Line 2: SERVICE:     NEWMACH  193
      :
```

6. XEDIT the file *resid* SQLDBN Q. Change the original database user ID to the new user ID. For example:

Before:

```
DBMACHID=SQLMACH,DCSSID=dcssid,DBNAME=dbname
```

After:

```
DBMACHID=NEWMACH,DCSSID=dcssid,DBNAME=dbname
```

7. XEDIT the file *resid* SQLFDEF Q. Change all the occurrences of the original database user ID to the new user ID. For example:

Before:

```

/**** FOLLOWING LINES FOR BDISK ****/
If var1='DEF' Then CP LINK SQLMACH 200 200 W
:
/**** FOLLOWING LINES FOR LOGDSK1 ****/
If var1='DEF' Then CP LINK SQLMACH 201 201 W
:
/**** FOLLOWING LINES FOR LOGDSK2 ****/
If var1='DEF' Then CP LINK SQLMACH 202 202 W
:
/**** FOLLOWING LINES FOR DDSK1 ****/
If var1='DEF' Then CP LINK SQLMACH 203 203 W
:
/**** FOLLOWING LINES FOR DDSK2 ****/
If var1='DEF' Then CP LINK SQLMACH 204 204 W
:
/**** FOLLOWING LINES FOR DDSK20 ****/
If var1='DEF' Then CP LINK SQLMACH 216 216 W
:

```

After:

```

/**** FOLLOWING LINES FOR BDISK ****/
If var1='DEF' Then CP LINK NEWMACH 200 200 W
:
/**** FOLLOWING LINES FOR LOGDSK1 ****/
If var1='DEF' Then CP LINK NEWMACH 201 201 W
:
/**** FOLLOWING LINES FOR LOGDSK2 ****/
If var1='DEF' Then CP LINK NEWMACH 202 202 W
:
/**** FOLLOWING LINES FOR DDSK1 ****/
If var1='DEF' Then CP LINK NEWMACH 203 203 W
:
/**** FOLLOWING LINES FOR DDSK2 ****/
If var1='DEF' Then CP LINK NEWMACH 204 204 W
:
/**** FOLLOWING LINES FOR DDSK20 ****/
If var1='DEF' Then CP LINK NEWMACH 216 216 W
:

```

8. If the database server name is going to be different, follow the steps described in “Changing the Server Name and Resource Identifier” on page 53 to change the server name and the resource ID.
9. Copy the database minidisks (Directory, log disks and dbextents) to the new *userid*. For details on moving the database minidisks, refer to the following sections in Chapter 9, “Making Backups and Recovering from Failures” on page 203:
 - Replacing a Minidisk Using DASD Dump Restore
 - Replacing a Database Minidisk
 - Replacing a Log Minidisk
10. VMSES/E considerations: To apply service to the new user ID, you must create a PPF (Product Parameter File) override to the 5648A70S \$PPF file. The PPF override must reflect the new database user ID and service and production minidisk address or SFS directory names. The \$PPF file resides on the VMSES/E Software Inventory disk (MAINT 51D). Therefore, when servicing

DB2 Server for VM, the files are copied to the correct database user ID. Refer to the *VM/ESA: VMSES/E Introduction and Reference* for more information on creating a PPF override.

Converting a Service Machine to a Database Machine

You may need at times to convert a service machine to a database machine if a database is required on that processor.

To convert this processor to a database machine, all you have to do is generate a database and make the appropriate VM directory changes.

For more information, see “Adding a Primary Database Machine” on page 287.

Changing the Server Name and Resource Identifier

Situations exist where you may want to change the application server name or the resource identifier. For example, you may want to change an application server name from 8 to 18 characters or have it conform to your naming conventions, or you may want to change a resource identifier name to a registered DRDA TPN.

The first character of the application server name must be an uppercase letter (A–Z), followed by alphanumeric characters. The name must be from 1 to 18 characters.

The following example shows how to rename an application server with a character resource identifier. In this example, the DB2 Server for VM production minidisk is assumed to be the Q-disk, and an application server with a DBNAME of *dbname1* and a RESID of *resid1* will be renamed to *dbname2 resid2*.

Notes:

1. In a DRDA2 environment, the database manager uses the RESID to create its own log name. Therefore, before changing the server name, ensure that any DRDA2 in-doubt agents have been resolved. Once the server name has changed, use the RESET CRR LOGNAMES command to clear old log names. When the database manager is next started with the initialization parameters PROTOCOL=AUTO and SYNCNT=Y and **Resynchronization Initialization occurs**, the DB2 Server for VM log status will be COLD.
2. If the RESID NAMES Q file is not on the production minidisk, use XEDIT to create the file. Create it with one entry that has the following three tags:

```
:nick  
:dbname  
:resid
```

1. Access the Q-disk with write capability.
2. Enter the CMS command:

```
FILELIST resid1 * Q
```

A list of files from the Q-disk is displayed. Three files are on the Q-disk for each application server:

- a. *resid1* SQLDBGEN
- b. *resid1* SQLDBN

c. *resid1* SQLFDEF

3. Rename the file name of each file:

```
RENAME / resid2 SQLDBGEN Q
RENAME / resid2 SQLDBN Q
RENAME / resid2 SQLFDEF Q
```

4. Edit *resid2* SQLDBN Q:

```
XEDIT resid2 SQLDBN Q
```

This file contains a statement similar to the following:

```
DBMACHID=Dbmachid,DCSSID=Dcssid,DBNAME=dbname1
```

Replace the DBNAME value (DBNAME=dbname1) with your new server name:

```
DBMACHID=Dbmachid,DCSSID=Dcssid,DBNAME=dbname2
```

5. Edit RESID NAMES Q:

```
XEDIT RESID NAMES Q
```

In this file, you see the :DBNAME and :RESID tags. Replace the resource identifier *resid1* with *resid2*, and *dbname1* with *dbname2*.

Before:

```
:nick.      :dbname.dbname1  :resid.resid1
```

After:

```
:nick.      :dbname.dbname2  :resid.resid2
```

6. If the old server name or *resid* (for example, TPN) is referenced by any CMS communication directory entries, you must update those directory entries.

After changing your application server name and resource identifier, ensure that users enter:

```
SQLINIT DB(dbname2)
```

to identify the application server to be accessed.

When you want to start the renamed application server, specify the new server name when you enter SQLSTART:

```
SQLSTART DB(dbname2)
```

The following example shows how to rename an application server, and how to change a resource identifier to a registered DRDA TPN. In this example, the DB2 Server for VM production minidisk is assumed to be the Q-disk, and an application server with a DBNAME of *dbname1* and a RESID of *resid1* will be renamed to *dbname2 07F6C4C2*. The RESID *07F6C4C2* represents the default DRDA TPN X'07F6C4C2'.

Note: If the RESID NAMES Q file is not on the production minidisk, use XEDIT to create the file. Create it with one entry that has the following three tags:

```
:nick
:dbname
:resid
```

1. Access the Q disk with write capability.
2. Enter the CMS command:


```
FILELIST resid1 * Q
```

A list of files from the Q-disk is displayed. Three files are on the Q-disk for each application server:

- a. resid1 SQLDBGEN
- b. resid1 SQLDBN
- c. resid1 SQLFDEF

3. Rename the file name of each file:

```
RENAME / 07F6C4C2 SQLDBGEN Q
RENAME / 07F6C4C2 SQLDBN Q
RENAME / 07F6C4C2 SQLFDEF Q
```

4. Edit 07F6C4C2 SQLDBN Q:

```
XEDIT 07F6C4C2 SQLDBN Q
```

This file contains a statement similar to the following:

```
DBMACHID=Dbmachid,DCSSID=Dcssid,DBNAME=dbname1
```

Replace the DBNAME value (DBNAME=dbname1) with your new server name:

```
DBMACHID=Dbmachid,DCSSID=Dcssid,DBNAME=dbname2
```

5. Edit RESID NAMES Q:

```
XEDIT RESID NAMES Q
```

In this file, you will see :DBNAME and :RESID tags. Replace the resource identifier resid1 with the hexadecimal value X'07F6C4C2' and *dbname1* with *dbname2*.

Before:

```
:nick.          :dbname.dbname1  :resid.resid1
```

After:

```
:nick.          :dbname.dbname2  :resid.X'07F6C4C2'
```

6. Enter a hexadecimal TPN value in the CMS communication directory by using the SET VERIFY ON HEX 1 72 command, and entering the hexadecimal digits.

Chapter 4. Planning for Operation of the Database Manager

Once the DB2 Server for VM code is installed and your database generated, the operator can start the application server so that users can access the databases and submit SQL statements. This chapter explains the planning tasks associated with starting, running, and stopping the application server. For information on the actual operator commands, see the *DB2 Server for VSE & VM Operation* manual.

Starting the Application Server

This section discusses the following topics:

- The database operator
- Multiple user mode initialization parameters
- Single user mode initialization parameters
- Tape support
- General file support
- Starting the application server in multiple user mode
- Running multiple user mode applications
- Starting the application server in single user mode
- Overriding initialization parameters
- Creating a parameter file

The Database Operator

Each database machine has its own operator console called the DB2 Server for VM operator console. The user who operates this console is referred to as the database operator.

When more than one database machine is active, there is more than one database operator console. With VM facilities, a single person can operate many database machines. For example, one person can operate many database machines by running the virtual machines disconnected. This common operator can reconnect to the various machines as needed.

Another way to have one operator is to use the VM Single Console Image Facility or the Programmable Operator Facility. These facilities allow the VM system operator to operate all the database machines. To learn more about the single console image facility and the programmable operator facility, refer to the *VM/ESA: CP Programming Services* or the *VM/ESA: Planning and Administration* manuals.

The database manager can operate in one of two modes: multiple user mode, or single user mode.

In *multiple user mode*, one or more users or applications concurrently access the same database. The database manager runs in a virtual machine while one or more applications run in other virtual machines. Users specify the database they want to access by running the SQLINIT EXEC. This EXEC establishes a default database for each user. For example, a user who first wants to access a database called TEST, then use ISQL, would enter:

```
SQLINIT DB(TEST)
ISQL
```

The initialization parameter SYSMODE=M defines this mode.

In *single user mode*, the database manager and only one application program run in the same virtual machine. The application server is started, the program name is passed as a parameter to the database manager, the application is run, and the application server terminates. The initialization parameter SYSMODE=S defines this mode.

Multiple User Mode Initialization Parameters

Figure 8 identifies the initialization parameters that apply when the database manager is operating in multiple user mode, and lists their defaults. A discussion of the appropriate settings for these parameters follows.

PARAMETER	DEFAULT	MINIMUM	MAXIMUM
ENVIRONMENT PARAMETERS			
DBNAME(name)	None	—	—
DCSSID()	SQLDBA	—	—
AMODE()	31	—	—
SYSMODE=M	M	—	—
STARTUP=W R U	W	—	—
PARMID=name	None	—	—
DBMODE=G L N	*IDENT Directory	—	—
	Entry	—	—
PROTOCOL=	SQLDS	—	—
SQLDS AUTO			
CHARNAME=name	INTERNATIONAL	—	—
ACCOUNT=D N	N	—	—
SYNCPNT=Y N	If PROTOCOL=AUTO, Y	—	—
DSPSTATS=nn	0	0	21
PERFORMANCE PARAMETERS			
NCUSERS=n	5	1	252
NPACKAGE=n	10	1	32766
NPACKPCT=n	30	0	100
NPAGBUF=n	10 + NCUSERS X 4	10	3500
NDIRBUF=n	NPAGBUF	10	28000
NLRBU=n	1000	10	583333
NLRBS=n	(2 X NCUSERS) + (NLRBU x NCUSERS)/2 + 10	larger of 50 or (2 x NCUSERS)	583333
DISPBIAS=n	7	1	10
NCSCANS=n	30	1	655
LTIMEOUT=n	0	0	99999
RECOVERY PARAMETERS			
LOGMODE=Y A L	Y	—	—
CHKINTVL=n	10	1	99999999
SLOGCUSH=n	90	11	90
ARCHPCT=n	80	10	99
SOSLEVEL=n	10	1	100
SERVICE PARAMETERS			
DSPLYDEV=L C B	L	—	—
DUMPTYPE=P F N	F	—	—
EXTEND=Y N	N	—	—
TRACDBSS=nnn...	000...	000...	222...
TRACRDS=nnnnnnn	0000000	0000000	2222222
TRACWUM=n	0	0	2
TRACDRRM=nnnn	0000	0000	2222
TRACDSC=nn	00	00	22
TRACCONV=n	0	0	2
TRACSTG=n	0	0	1
TRACEBUF=n	0	0	99999

Figure 8. Multiple User Mode Initialization Parameters

Environment Parameters

DBNAME

A database machine can own more than one database. When starting the application server, specify the name of the database that is to be accessed by the database machine. Note that DBNAME is not specified in the parameter list of the SQLSTART command as an initialization parameter. Figure 9 on page 60 shows the DBNAME parameter specified correctly.

DCSSID

This parameter specifies the name of the bootstrap package to be used. It is not needed if saved segments are not being used. For more information on saved segments and specifying the DCSSID parameter, see Chapter 8, “Saved Segments” on page 181. Figure 9 shows the DCSSID parameter specified correctly. For more information on the use of this parameter, see “Starting the Application Server in Multiple User Mode” on page 83.

AMODE

This parameter specifies the type of addressing the database manager runs in: 31-bit addressing or 24-bit addressing. Note that AMODE is not specified in the parameter list of the SQLSTART EXEC as an initialization parameter. Figure 9 shows the AMODE parameter specified correctly. For more information on the use of this parameter, see “Starting the Application Server in Multiple User Mode” on page 83.

SYSMODE

This parameter is used to specify either single(S) or multiple(M) user mode. Set it to M to initialize the database manager for multiple user mode operation. This is the default mode. You will NOT normally specify this parameter as SQL EXECs set this parameter for you automatically.

STARTUP

This parameter specifies how the database will be started:

- Most of the time let STARTUP default to W (warm start).
- Use STARTUP=R (restore) to restart the application server and restore the database from an archive tape file.
- Specify STARTUP=U (user restore) if you have archived and restored the database with user facilities.

For more information, see “Restoring the Database” on page 224.

PARMID

This parameter can be used to specify a CMS file containing the values for the other initialization parameters. Application program parameters (user parameters) cannot be included. Specify only the file name for PARMID. The database manager assumes that the file type is SQLPARM and the file mode is *. The * tells CMS to search all accessed minidisks (A to Z). Figure 9 shows an example startup that uses the PARMID parameter.

```
SQLSTART DBNAME(SQLDBA) DCSSID(MYBOOT) AMODE(31) PARM(PARMID=WARM1,LOGMODE=A)
```

Figure 9. Starting in Multiple User Mode with a CMS File

DBMODE

This parameter identifies the database name as a LOCAL resource (DBMODE=L), a GLOBAL resource (DBMODE=G), or non-APPC/VM (DBMODE=N) for a particular session. If the DBMODE parameter is not specified, the resource authorization specified in the VM directory of the database machine is used. Consider the following when specifying the DBMODE parameter:

- If you specify DBMODE=L or G to run SQLSTART and the database machine directory does not contain the control statement IUCV with parameters GLOBAL or LOCAL, the SQLSTART EXEC fails.
- When DBMODE=G is specified, and the IUCV *IDENT directory entry does not allow that resource name to be identified as a GLOBAL resource, the application server ends the startup.
- If DBMODE=L is specified, the application server is identified as a LOCAL resource even if the IUCV directory entry specifies that the resource is GLOBAL. Specify this parameter to restrict access to the application server to users on the local processor.
- If DBMODE=N is specified, no *IDENT directory entries are required, because the database machine uses IUCV instead of APPC/VM. You should use DBMODE=N if you have just migrated to a VM/ESA operating system, and you do not want to make directory changes yet. For more information about DBMODE=N, see “Delaying the Directory and Database Name Changes” on page 44.

For more information about directory entries, see “VM Directory Control Statements” on page 146.

PROTOCOL

This parameter specifies the types of protocols that the application server can handle. It has two options on the SQLSTART EXEC: SQLDS and AUTO.

When PROTOCOL=SQLDS is specified, the DB2 Server for VM application server allows access from DB2 Server for VM application requesters **only**. These application requesters and application server can be in either a local or remote environment. This option is the default.

When PROTOCOL=AUTO is specified, the DB2 Server for VM application server allows access from DB2 Server for VM and non-DB2 Server for VM application requesters. This parameter can only be specified if the DRDA code has been installed. It is used with the SYNCNT parameter to control the DRDA environment. For more information, see the description of the SYNCNT parameter and see Chapter 15, “Using a DRDA Environment” on page 411.

On the application server, the PROTOCOL parameter is specified using the SQLSTART EXEC. On the application requester, the SQLINIT EXEC also has a PROTOCOL parameter. When a connection is made between the application requester and the application server, the combination specified by these parameters determines whether the DRDA protocol or the SQLDS protocol is to be used for that connection.

You should be aware of the performance impacts of the chosen protocol. For a detailed explanation on protocols, see the *DB2 Server for VSE & VM Performance Tuning Handbook*.

For a list of restrictions when using the DRDA protocol, see Appendix H, “DRDA Considerations” on page 499. For information on the SQLINIT EXEC, see the *DB2 Server for VM Database Administration* manual.

CHARNAME

This section discusses the following:

- Character set considerations at startup
- National language considerations at startup.

Character Set Considerations at Startup: Use the CHARNAME parameter to specify the CCSIDs to be used as the application server defaults. The default CCSIDs determine the character sets and code pages to be used to interpret statements and return results.

The valid CHARNAME values you can specify are ENGLISH (CCSID=37), INTERNATIONAL (CCSID=500), and all the values that are in the CHARNAME column of the SYSTEM.SYSCCSIDS catalog table.

The database manager obtains the CCSIDs associated with the CHARNAME by looking up the row of the SYSTEM.SYSCCSIDS catalog table where the CHARNAME column matches the CHARNAME parameter. It also obtains the classification and translation tables associated with the CHARNAME by looking up the row of the SYSTEM.SYSCHARSETS catalog table where the NAME column matches the CHARNAME parameter. The classification table is used to identify valid characters in identifiers. The translation table is used to indicate how to fold ordinary lowercase identifiers to uppercase.

For CHARNAMEs ENGLISH and INTERNATIONAL, their CCSID values, the classification table and the translation table are stored internally. The rows in SYSTEM.SYSCCSIDS and SYSTEM.SYSCHARSETS for these CHARNAMEs are for reference purposes only and are not used by the database manager.

During startup, if you do not specify the CHARNAME parameter, the application server uses the same CHARNAME that was used the last time it was started. The values stored in the rows where SQLOPTION equals CHARNAME, CCSIDSBBCS, CCSIDMIXED, and CCSIDGRAPHIC are for reference purposes only. They reflect the current values associated with the system. The only way to change the default values is by starting the application server with a different CHARNAME parameter. Any updates to the values in the SYSTEM.SYSOPTIONS table are ignored during startup.

Note: The database manager determines the current default CHARNAME from the CCSID attribute of the CNAME character column in the SYSTEM.SYSCOLUMNS catalog table. If this value is null, then 37 is used (a CCSID of 37 corresponds to a CHARNAME of ENGLISH). The database manager uses the CCSID value to locate the corresponding row in the ARISCCS MACRO file to obtain the associated CHARNAME. The value in the CHARNAME column of this row is the current application server default CHARNAME.

When you specify a value for the CHARNAME parameter that is different from the current application server default CHARNAME, you are prompted to choose whether or not you want to change the application server default CHARNAME. If you specify YES and have supplied a valid CHARNAME value, the database manager updates the application server default values for CHARNAME, CCSIDSBBCS, CCSIDMIXED, and CCSIDGRAPHIC. It also modifies the CCSID attribute of all character columns that are part of the catalog tables to the application server default CCSID. The CCSID attribute of character columns that

are not part of the catalog tables are not modified. If the value for CCSIDMIXED is not zero, this value is used as the application server default CCSID. If the value for CCSIDMIXED is zero, then the application server default CCSID is the value of CCSIDSBSCS.

Note that the tables which have their CCSID modified when the CHARNAME is changed include:

- All tables created by SYSTEM
- The following tables created by SQLDBA:
 - SQLDBA.ROUTINE
 - SQLDBA.STORED QUERIES
 - SQLDBA.SYSLANGUAGE
 - SQLDBA.SYSTEXT2
 - SQLDBA.SYSUSERLIST

When a CHARNAME is changed, the following should be considered:

1. The FIPS Flagger package **must** be reloaded by using the ARISDBMA EXEC. Failure to do this can cause SQLCODE=-931 (SQLSTATE=58004). This will render the agent reporting the SQLCODE error unable to preprocess packages until the application server is started. Once the FIPS Flagger package is reloaded or reprocessed, this error will not occur.
2. All views which are dependent on the tables that had their CCSID modified must be dropped and recreated.

The following query lists all such view packages:

```
SELECT CREATOR, TNAME, PLABEL
FROM SYSTEM.SYSACCESS
WHERE TABTYPE = 'V'
AND VALID = 'N'
```

This query is useful in that owners of affected views can be notified to drop and recreate their view before they try and use the view and get an error (SQLCODE=-835, SQLSTATE=56049, with SQLERRD1 set to -833).

3. All packages which are dependent on the tables that had their CCSID modified must be dropped and recreated.

The following query lists all such packages:

```
SELECT CREATOR, TNAME, PLABEL
FROM SYSTEM.SYSACCESS
WHERE TABTYPE = 'X'
AND VALID = 'N'
```

This query is useful in that owners of affected packages can be notified to rebind the packages instead of having them dynamically reprocessed at run time. The DBS utility REBIND PACKAGE command can be used to rebind the packages listed.

4. The ISQL package (SQLDBA.ARIISQL) and DBS utility package (SQLDBA.ARIDSQL) can be reloaded and recreated using the ARISDBMA EXEC. If this is not done, the first time these packages are used, they will be dynamically reprocessed.

To check if all the above activities have been done, run the following query:

```

SELECT CREATOR, TNAME, PLABEL
FROM SYSTEM.SYSACCESS
WHERE VALID = 'N'

```

If there are no rows found, all packages have been either recreated, reloaded, rebound or dynamically reprocessed and the VALID column value for the package in SYSTEM.SYSACCESS has been changed to “Y.”

Note that CCSID conversion of the data in catalog tables does not occur: only the CCSID attribute of the columns is modified. If you change the application server default CHARNAME, system objects of the character data type (for example, table names and column names) stored in the catalog may be displayed differently. The reason for this is that a code point may represent different characters in different code pages.

If you want to change the application server default CHARNAME, the default will not be changed if:

- You specify an invalid value for the CHARNAME parameter
- An error occurs in the verification of the
 - New CHARNAME CCSID values
 - Classification table
 - Translation table.

When the application server is started, it records the application server default values for CHARNAME, CCSIDSBBCS, CCSIDMIXED, and CCSIDGRAPHIC in the SYSTEM.SYSOPTIONS catalog table. To obtain these values, you can query the table. For example, to determine the name of the character set that is currently in use, issue:

```

SELECT VALUE
FROM SYSTEM.SYSOPTIONS
WHERE SQLOPTION = 'CHARNAME'

```

For more information about character sets, see Chapter 13, “Choosing a National Language and Defining Character Sets” on page 319.

National Language Considerations at Startup: You can use the SET LANGUAGE command from the operator console to choose a national language so that DB2 Server for VM messages can be received in the selected language. For more information see “National Language Support for Messages and HELP Text” on page 348.

ACCOUNT

This parameter enables the accounting facility. When ACCOUNT=D is specified, accounting records are generated and directed to the VM system accounting file. If the default value of ACCOUNT=N is specified, accounting information is not generated.

For a complete description of the accounting facility, see Chapter 11, “Using the Accounting Facility” on page 261.

SYNCPNT

This parameter specifies whether or not a sync point manager (SPM) will be used to coordinate DRDA2 DUOW two-phase commit and resynchronization activity. It is only meaningful when `PROTOCOL=AUTO`.

If Y is specified, the server will use a sync point manager, if possible, to coordinate two-phase commits and resynchronization activity. If N is specified, the server will not use an SPM to perform two-phase commits. If N is specified, the database manager is limited to multi-read, single-write distributed units of work and it can be the single write site. If Y is specified, but the database manager finds that a sync point manager is *not* available, then the server will operate as if N was specified.

The default is `SYNCPNT=Y`, if `PROTOCOL=AUTO`.

DSPSTATS

This two digit parameter specifies what information is displayed and what level of detail is displayed. If 0 is specified, nothing is displayed. If 1 is specified, the minimum information is displayed. If 2 is specified, more detail is displayed. The positional digits correspond to the following informational displays: the first is checkpoint performance information and the second is counter information to be displayed at system shutdown.

If the first option is 1, then format 1 of message ARI2052I is displayed every time a checkpoint occurs. This is useful in determining how often checkpoints occur. If the first option is 2, then format 2 of message ARI2052I is displayed every time a checkpoint occurs. This is useful in determining if checkpoint processing is causing a performance problem.

If the second option is 1, then the "COUNTER **" operator command is issued just before the application server is shutdown. This is useful for performance tuning. If the dataspaces feature is being used, "COUNTER POOL **" command is also issued.

The SET command changes the value of this parameter without having to stop and restart the application server. For more information on the SET operator command, see the *DB2 Server for VSE & VM Operation* manual.

SECALVER

This parameter determines if the application server will accept users that have already been verified by another system. If `SECALVER=Y`, verified users will be accepted. The requester only needs to send a user ID to be validated. If `SECALVER=N`, verified users will not be accepted. The requester must send a user ID and password to be verified.

Note: This parameter is only used when validating users are connecting via TCP/IP or when users send the ACCSEC and SECCHK DRDA datastreams in their connect request.

SECTYPE

This parameter determines if the application server will validate a user ID and password for connect authority using an external security manager or by checking the DB2 SYSUSERAUTH catalog table. If `SECTYPE=ESM` an external security manager will be used to validate the user ID and password. The external security manager must support the RACROUTE application programming interface. If

SECTYPE=DB2, the user ID and password are validated by checking the SYSUSERAUTH catalog table.

Note: This parameter is only used when validating users are connecting via TCP/IP or when users send the ACCSEC and SECCHK DRDA datastreams in their connect request.

TCPPOPT

This parameter specifies the TCP/IP port number that the application server will use to listen for incoming TCP/IP connect requests.

If this parameter is not specified, TCP/IP support will be initialized and the ETC SERVICES file on the TCP/IP client disk will be searched to determine the port number that the application server will use.

If this parameter is specified with a non-zero value, TCP/IP support will be initialized and the value specified will be used as the port number that the application server will use.

If this parameter is specified with a value of 0, TCP/IP support will not be initialized.

Performance Parameters

NCUSERS

This parameter defines the maximum number of *real* agents that the database manager can actively handle at any one time, limiting the number of users that can be supported by the database manager. The value of NCUSERS is usually less than the number of connected users anticipated, because not all users will be accessing data at the same time. This value directly affects the size of the virtual machine required.

The number of NCUSERS is limited because some static agent storage for each real agent is obtained below 16 megabytes. See Table 36 on page 444.

Figure 10 provides guidelines for setting the NCUSERS parameter. Because these are only guidelines, you should modify them to concur with the activity on your system. For additional information, see the *DB2 Server for VSE & VM Performance Tuning Handbook*.

<p>NCUSERS= 1 for each 1-2 users of ISQL (or other query products) + 1 for each 1-25 non-ISQL users (variable on transaction workload) + 1 for each 2-5 application program developers</p>
--

Figure 10. Guidelines for the NCUSERS Parameter

If you have application programs that maintain multiple logical units of work in separate CMS work units, each additional work unit used by an application at one time must be counted as an additional user.

Each ISQL user can generate a high level of system activity. If you set NCUSERS so that all ISQL users can be active at the same time (NCUSERS=number of ISQL users), you minimize the time that any one user must wait for services. However, if this number is large, it may cause the database manager to be overloaded. To

prevent this, you should also set the MAXCONN parameter of the VM OPTION directory control statement, which limits the number of users and the number of DASDs that a given virtual machine can access. For information on this parameter, see “Inter-Machine Communications” on page 100.

Application developers typically do a considerable amount of other activity (such as CMS file editing or output scanning). These users require less service from the database manager, so NCUSERS can be lowered accordingly.

If you are using VSE guest sharing, the NCUSERS of the VM database machine should be increased by the number required for the VSE guest. The demand for services from CICS transaction processing can vary widely, depending on the nature of the transactions.

The demand for services from batch application programs can also vary considerably. If you have online or interactive activity on the database manager, consider limiting the amount of concurrent SQL batch processing.

Note: When the application server is started, there may be one or more in-doubt logical units of work (LUWs). The value of NCUSERS must be large enough to handle these. When they have been resolved, the DB2 Server for VM agent structures are used to handle new users. The creation and use of agent structures for resolving in-doubt LUWs takes precedence over all new user logical units of work. For more information about in-doubt LUWs, see “Resolving In-Doubt Transactions” on page 121.

NPACKAGE

This parameter defines the maximum number of packages in an LUW, and together with the value specified for NCUSERS, determines the size of the *package cache*. The size of the package cache limits the number of packages that can be present in storage simultaneously. (Package cache size = NPackages x NCUSERS.) The default value of NPackages is 10, and that for NCUSERS is 5, giving a default package cache of 50, allowing 50 packages to be present in storage simultaneously.

In general, increasing the size of the package cache improves performance of the database manager. However, do not increase it to the point where system paging becomes too great. For more information, see the *DB2 Server for VSE & VM Performance Tuning Handbook*.

NPACKPCT

This parameter defines the percentage of the package cache that is used in the calculation of the package cache *threshold*. The size of the threshold determines the number of loaded packages that are kept in storage at the end of an LUW. (Threshold = NPACKPCT percent of package cache.) If the threshold is exceeded, the loaded packages are freed and returned to the package cache.

The default values for NPACKPCT and the package cache are 30 and 50 respectively, giving a threshold of 15. In general, increasing the size of the threshold improves performance. For more information, see the *DB2 Server for VSE & VM Performance Tuning Handbook*.

NPAGBUF

This parameter specifies the number of 4096-byte data pages kept in storage buffers at one time. The number of data buffers you want depends on the number of active users and the nature of their request. The default for NPAGBUF assumes an average of four buffer pages for each potentially active user ($NCUSERS \times 4$), plus ten buffer pages for the buffering of catalog and log information.

In general, increasing NPAGBUF improves the performance of the database manager. However, increasing it also requires an increase in the size of the database machine. Also -- and more importantly -- it can cause an increase in the paging rate of the system. It is more efficient to let the database manager do more I/O operations than it is to let the system do more paging; database I/O operations are overlapped whereas system paging operations are not. Therefore do not increase NPAGBUF to the point where system paging becomes too great.

For more information about NPAGBUF, see the *DB2 Server for VM Diagnosis Guide and Reference* manual.

NDIRBUF

This parameter determines the number of 512-byte directory pages to be kept in storage. Increasing it reduces the number of I/O operations. Again, bigger is better, until you either run out of virtual storage or cause too much system paging. Each directory page addresses 128 data pages.

When you set NPAGBUF and NDIRBUF, you have to choose how to split buffer space between data pages and directory pages. At least initially, you should set them to the same value. Issue the COUNTER commands to see the actual I/O activity; then adjust NPAGBUF and NDIRBUF.

For more information about NDIRBUF, see the *DB2 Server for VM Diagnosis Guide and Reference* manual.

NLRBU and NLRBS

NLRBU specifies the maximum number of lock request blocks allowed for one active user, while NLRBS specifies the number allowed for *all* active users. (Usually, two lock request blocks are used for every lock that a user holds.)

The database manager can perform lock escalations, increasing the granularity of data being locked from either row or page level to dbspace level. In general, you only need to change the default values of NLRBU and NLRBS if contention problems occur. Increasing them reduces the number of lock escalations performed by the database manager.

When either the NLRBU limit for a user is reached or the NLRBS limit is approached, lock escalation occurs. This results in fewer locks being required, and lock request blocks being freed. This in turn reduces the opportunities to share data. For example, when locking is done at a row level, many users may be updating the same dbspace at the same time. When it is escalated to the dbspace level, only one user can update rows in that dbspace. Everyone else must wait until that person's update is committed or rolled back.

Escalation can also cause deadlocks. A deadlock occurs when two or more LUWs are in wait states and dependent on the completion of LUWs that are also in wait states. For example, suppose two users are updating tables in a dbspace. When

the lock size is escalated to a dbspace level, both users can be locked out, with each waiting for the other to complete an LUW. The database manager resolves situations like these by rolling back the newest LUW. For more about locking, see the *DB2 Server for VM Application Programming* manual.

If the default values for NCUSERS (5) and NLRBU (1000) are used, the database manager defines 2520 lock request blocks, each of which requires 24 bytes; 60480 bytes of virtual storage are required for lock request blocks. With these defaults, one application could use 1000 lock request blocks and four other applications could simultaneously use an average of 370 lock request blocks each, before causing an escalation.

Even though two lock request blocks are needed for each lock, the default values allow a large number of locks for each application. With the defaults, one application could use 500 locks while four other applications use an average of 185 locks each.

You should use the NLRBU and NLRBS default values at first, and increase them if users either are experiencing delays when they access the database manager, or if they are receiving SQLCODEs of -911, -912, or -915 (rollbacks that occur because of deadlock, insufficient lock request blocks for the database manager, or insufficient lock request blocks for a user application, respectively).

Note: These SQLCODEs may also be received during preprocessing, as the locks are required then as well.

To test the frequency of lock escalations and of deadlocks, use the COUNTER operator command. Specify both the ESCALATE and the LOCKLMT counters to get the number of successful escalations and the number of unsuccessful escalation attempts respectively. (An escalation can fail if the LUW that reached the lock limit is rolled back because of a deadlock, or if a sufficient number of lock request blocks cannot be freed.) For example, suppose the operator issues the command COUNTER ESCALATE LOCKLMT a few times a day and normally receives results in the range of 10 to 150 for ESCALATE, and 0 to 5 for LOCKLMT. If, one day, the results are 428 for ESCALATE and 23 for LOCKLMT, a locking problem would be indicated.

In addition, the SHOW LOCK MATRIX command can be used to display information about lock request block usage to determine whether unexpected delays are caused by locking; to monitor how the database manager is using lock request blocks; and to determine the lock request blocks required for a single application or for a run of a preprocessor.

One of the values displayed is called MAX USED BY LUW: the maximum number of lock request blocks used by any one application during an LUW. (When any LUW starts to exceed NLRBU and the escalation process occurs, MAX USED BY LUW is set to zero.) All this information can help you determine the required values for NLRBU and NLRBS.

To establish the lock request block requirements for running a preprocessor, or for an application that is causing contention problems:

1. Start the application server in multiple user mode with `NCUSERS=1`, `NLRBU` about five times its current setting, and `NLRBS` set to the same value as `NLRBU`.
2. Start the application and allow it to complete processing.
3. Verify that no escalation occurred by displaying the `ESCALATE` and `LOCKLMT` counters. If no escalation occurred, `MAX USED BY LUW` will show the number of lock request blocks required.
4. If an escalation did occur, set `NLRBU` to a value greater than or equal to `MAX USED BY LUW`, then start the application server again, and rerun the application.

If necessary, reset `NLRBS`. For example, suppose `NLRBU` is set to 1100, and two users will run their applications -- each requiring 1100 lock request blocks -- at the same time. Also assume that any other application requires about 500 lock request blocks. If `NCUSERS` is 5, then set `NLRBS` to at least 3700 (1100 for each of two applications and 500 for each of three additional applications).

If an application requires more lock request blocks than you have virtual storage for, you should consider the following alternatives:

- Use either the SQL `ALTER DBSPACE` or the SQL `LOCK` statement to change the locking level of the `dbspace` used by the application. The `ALTER` statement permanently changes the locking level for all applications, while the `LOCK` statement can be inserted into an application, and used to change the locking level only when that application runs. The `LOCK` statement is the preferred way to temporarily modify the locking level, because it involves no update to the catalog tables.
- Consider changing the application: perhaps it is holding locks longer than necessary. Additional SQL `COMMIT WORK` statements in the application may necessitate fewer locks.
- Consider running the application by itself: either in single user mode, where no locking is required, or in multiple user mode with a reduced `NCUSERS` and with `NLRBU` and `NLRBS` set as required.

For more information about locking problems and how to solve them, see the *DB2 Server for VM Diagnosis Guide and Reference* manual.

DISPBIAS

This parameter determines how the dispatcher selects the order in which agents get serviced by the database manager. To set it, you need to understand how the dispatcher works. Only one agent at a time can be serviced; the other agents wait in a queue. Within this queue, agents are prioritized according to their estimated resource consumption: those estimated to consume the least are placed at the top, while those estimated to consume the most are placed at the bottom.

When the active agent returns to the dispatcher, the next agent at the top of the queue is dispatched. Every time an agent is dispatched, the database manager reevaluates the priority of the remaining agents, and requeues them according to their new priorities.

A pure priority dispatcher can present some problems, however. If many short-running LUWs are present, the longer-running ones may never get serviced:

they are always at the bottom of the queue. To avoid this problem, *fair-share auditing* is used, whereby all the agents in the queue are checked periodically to see if they are receiving adequate service. When one is found that is not, its priority is changed and it is moved to the top of the queue.

If fair-share auditing is done frequently, the dispatcher tends to operate more like a *round-robin* dispatcher: agents get equal service because those at the bottom of the queue get bumped to the top more frequently. If it is done infrequently, the dispatcher tends to operate more like a *priority* dispatcher: agents get prioritized service because long-running agents are forced to wait at the bottom of the queue longer. (Eventually, fair-share auditing causes these agents to get service.)

The DISPBIAS parameter determines how often fair-share auditing is done. When it is set low (near 1), fair-share auditing is done frequently, and the dispatcher operates more in round-robin mode. When it is set high (near 10), fair-share auditing is done infrequently, and the dispatcher operates more in priority mode.

Initially, you should use the DISPBIAS default of 7. If your long-running LUWs are getting poor service, you may want to use a lower value; if your ISQL users are often waiting for long-running applications to complete, you may want to use a higher value. You can use the SET operator command to change the value of DISPBIAS without having to stop and restart the application server. See the *DB2 Server for VSE & VM Operation* manual for more information on the SET operator command.

Note: Any changes you make using the SET command are only in effect while the application server is running. If you stop and restart the application server, it will use the settings you specified in the startup procedure.

You may be tempted to set DISPBIAS to 10 to get good response time for ISQL users. Keep in mind, however, that a long-running LUW can hold a large number of locks. If other users are waiting for those locks, they must wait until the application frees them. If the application is waiting at the bottom of the queue, everyone is waiting. In this situation, you would want to have fair-share auditing occur more frequently, so the long-running unit can free the resources it has locked. The default of seven represents a balance between the interests of long-running and short-running LUWs.

NCSCANS

This parameter determines the number of internal control scan blocks kept for accessing tables and indexes. These blocks can vary in size and number depending on the type of query being performed. This discussion is concerned with long-running requests that might be queries or database change operations.

Scan control blocks contain positioning information related to a query. The positioning information can result from a user-defined cursor or by an internal cursor created by RDS. If an index is involved in the query, the size of the scan control block depends on the key size for that index. An average scan control block is assumed to be 50 bytes (32 bytes for control information, and an average key length of 18 bytes).

The maximum table size to hold the scan control block entries for each agent is 32 kilobytes (32768 bytes). This can contain 655 entries of 50-byte scan control blocks, which in general, is enough to support 255 user-declared cursors. If, however, the key lengths for indexes are long, the scan table supports fewer user

cursors. For example, if the key length for a given index associated with a cursor is 255 bytes, an entry would require 287 (255 + 32) bytes, and the maximum number of cursors possible using that index would be 114 (32 kilobytes divided by 287). That number would be reduced if the DB2 Server for VM requests caused internal cursors to be created. Internal cursors are always smaller than 50 bytes, and cannot use index keys.

If you have many complex requests, you may have to increase NCSCANS. If it is not set to a high enough value, users will get SQLCODE -522. For information on the virtual storage used by NCSCANS, see "Initial Virtual Storage Requirements of Components" on page 443.

LTIMEOUT

This parameter specifies a general lock wait timeout period for any SQL application, and especially as the way to avoid global deadlocks for DUOW applications.

The range of the LTIMEOUT value is 0 to 99999 seconds. The value of zero indicates that no lock timeout should be enforced for agents connected to this database. This is the default value for a database.

A nonzero lock timeout value will cause any agents waiting for a lock to have their current transaction rolled back when the lock timeout period has expired. The agent will notify the application that a lock timeout has occurred with SQLCODE -911 (SQLSTATE 40001). A reason code will be returned to indicate whether it is a deadlock or lock timeout situation (reason code 2 for a deadlock situation and reason code 68 for a lock timeout situation). The lock timeout period begins at the moment an agent requests a lock on any database resource. The full lock timeout period is allowed for each lock request.

The lock timeout control parameter should be adjusted in those environments where lock contention between applications has started to affect the desired performance and concurrency levels.

If a lock timeout is required for your environment, it is recommended that your starting value be equivalent to the maximum period of time that you want an application to wait for a lock.

Note: The LTIMEOUT parameter is changed through the SET operator command. The timeout value will affect any users currently in LOCK WAIT. If a user has been in a LOCK WAIT for 100 seconds and the value of LTIMEOUT is set to a value less than 100, that user will receive a timeout. For more information on the SET operator command, see the *DB2 Server for VSE & VM Operation* manual.

If lock timeout control is activated, you should ensure that all applications recognize and can handle the -911 SQLCODE that may be received as the result of a lock timeout initiated rollback.

Note: New units of work that are waiting to begin because a log archive is running or is scheduled to run are in a lock wait. The SHOW LOCK WANTLOCK operator command shows these units of work waiting to acquire an IX lock on the database. Because log archives can potentially take a significant amount of time to complete, units of work in this particular type of lock wait are ignored by the lock timeout function.

PROCMXAB

This parameter specifies the number of times a stored procedure is allowed to terminate abnormally, after which a STOP PROC ACTION REJECT is performed against the procedure and all subsequent SQL CALL statements for that procedure are rejected. Note that a timeout that occurs while waiting for a stored procedure server to be assigned for an SQL CALL statement is not included in this count.

PROCMXAB must be an integer between 0 and 255. The default, 0, means that the first abend of a stored procedure causes SQL CALLs to that procedure to be rejected. For production systems, you should accept the default.

PTIMEOUT

This parameter specifies:

- The number of seconds before DB2 Server for VSE & VM ceases to wait for an SQL CALL to be assigned to a stored procedure server. If the PTIMEOUT interval expires, the SQL CALL statement fails.
- The number of seconds before DB2 Server for VSE & VM ceases to wait for the START PSERVER command to complete. If the PTIMEOUT interval expires, a message is displayed and the START PSERVER command terminates.

The default for PTIMEOUT is 180.

Recovery Parameters

LOGMODE

This parameter determines whether archives will be taken for the database and the log. Specify LOGMODE=A to maintain an archive of the database, LOGMODE=L to maintain an archive of the log, and LOGMODE=Y if you want logging but do not want the log archived.

LOGMODE=A allows you to restore the database and apply the current log. LOGMODE=L allows you to maintain a database archive as well as log archives. The database archive followed by the log archives are applied during restore, then the current log is applied.

Use LOGMODE=A or L if it is important to protect the database against media (DASD) failures; otherwise use LOGMODE=Y.

Note: Each sequence of log archives must be preceded by a database archive, so if you use LOGMODE=L, you must occasionally take a database archive. You do not need to switch to LOGMODE=A to do so.

For more information on LOGMODE, see “Choosing a Log Mode” on page 208.

CHKINTVL

This parameter determines how often a checkpoint is taken. A *checkpoint* is an internal operation in which data and status information is written to permanent (DASD) storage, and a summary status record is written to the log. A checkpoint causes two important events:

- Storage pool space is freed.

As updates to data occur, duplicate copies of changed data pages are maintained. These copies (called *shadow pages*) are kept in the storage pools of the pages that were changed. A checkpoint frees the shadow pages, and thereby frees the storage pool space where they are kept.

- Log space may be freed.

If LOGMODE=Y, a checkpoint typically frees log space by moving the logical beginning of the log forward to the beginning of the oldest LUW still active at the time of the checkpoint. If LOGMODE=A or L, log space is only freed when an archive is taken; not on every checkpoint.

Checkpoints are taken periodically; however, by the time one is taken, there may be a large amount of data to be committed. If a failure should occur before it is committed, much processing may need to be redone after the database is restored.

The CHKINTVL parameter lets you take checkpoints at predetermined intervals. Its value is specified in terms of the number of log pages written between checkpoints. You can use the SET operator command to change the value of CHKINTVL without having to stop and restart the application server. See the *DB2 Server for VSE & VM Operation* manual for more information on the SET operator command.

Note: Any changes you make using the SET command are only in effect while the application server is running. If you stop and restart the application server, it will use the settings you specified in the startup procedure.

By setting it low, you minimize the risk of filling the log or storage pools. However, because checkpoints are time-consuming operations that suspend SQL processing until they are completed, they should be taken infrequently. For more information on setting CHKINTVL, see the *DB2 Server for VSE & VM Performance Tuning Handbook*.

SLOGCUSH

This parameter defines the point at which the log cushion is entered and log-full processing begins. Its value is expressed in terms of the percentage of the log size. The default of 90 means that when the log is 90% full, log-full processing will be initiated.

In log-full processing, the oldest active LUWs are rolled back until enough log space is freed to bring the percentage of the log in use below the SLOGCUSH level. Ideally, checkpoints and archiving would continually free log space so that the log would never reach the SLOGCUSH level.

If the log should become 100% full, the database manager would end abnormally, so you should set SLOGCUSH to a value that allows log-full processing to take effect (free some log space) before this happens. If the database manager is ending with log-full conditions, you may want to lower the SLOGCUSH value or increase the size of your log minidisks.

ARCHPCT

This parameter can be used to define a point at which an archive is automatically initiated. It is used only when LOGMODE=A or L is specified. Like SLOGCUSH, its value is expressed in terms of a percentage of the log.

Archives free up log space; however, they take some time to complete. If the SLOGCUSH value is reached during an online archive operation, all SQL

processing is suspended until the archive is done. For this reason, it is best to ensure that archives are initiated in time to finish before the log fills to the SLOGCUSH percentage. This is done by setting the value of ARCHPCT lower than the value of SLOGCUSH.

When the log becomes full to the ARCHPCT value, a message is issued to the database machine operator to mount an archive tape and identify the virtual address (*cuu*) of the tape drive. The database manager then takes a database or log archive depending on whether you have LOGMODE set to A (database) or L (log).

If LOGMODE=L, the operator can also direct the log archive to disk. For more information, see “Log Archiving to Disk” on page 219.

Normally, the operator explicitly archives the database or the log before the ARCHPCT value is reached, by issuing one of the archive commands. If the ARCHPCT is reached, meaning that the log is almost full, the action that the database manager takes depends on the LOGMODE that is in effect. See Table 6 for a summary of these actions.

<i>Table 6. Summary of Activity When ARCHPCT Level Is Reached</i>	
LOGMODE Parameter	Activity When ARCHPCT is Reached
A	An operator message is issued that requests a database archive.
L	An operator message is issued that requests a log archive.
Y	Because the log cannot be archived, the value for ARCHPCT is ignored. When the log is full it wraps. If an LUW spans the entire log, a ROLLBACK WORK is forced for that LUW.

Note: To see how full the log is, you can issue the SHOW LOG command. For a description of this command, see the *DB2 Server for VSE & VM Operation* manual.

SOSLEVEL

This parameter defines the storage cushion for storage pools. Its value is expressed as a percentage of space remaining in a storage pool. In multiple user mode processing (and single user mode processing where LOGMODE is not N), if any storage pool gets full to the point where only the SOSLEVEL percentage of storage pool pages is still free, a checkpoint is taken to free any shadow pages in use.

If, following this, only enough pages are freed to bring the number of free pages just above the SOSLEVEL, frequent checkpointing could occur. For more information, see the *DB2 Server for VM Diagnosis Guide and Reference* manual. If, however, the number of free storage pool pages is still at or below SOSLEVEL, message ARI020I is issued once to inform the user that the number of free pages left in the storage pool is fewer than the SOSLEVEL. This message is also issued once in single user mode with LOGMODE=N, but no checkpoint is taken.

Attention: If message ARI020I is received, it indicates some action may be needed to prevent imminent filling of the storage pool.

One possible action is to stop the application server and extend that storage pool with the SQLADBEX EXEC. However, you can remedy the situation without

stopping the application server if you have set SOSLEVEL high enough to give you adequate warning. When the message is received, proceed to remove unneeded data from the storage pool, either by dropping dbspaces or tables, or by reorganizing the data with a smaller percentage of free space for each page. In order to do this, you must have adequate warning to schedule the necessary processing.

Service Parameters

DUMPTYPE

This parameter defines whether or not dumps are to be taken, and the amount of information to be dumped if they are.

DUMPTYPE=N indicates that a dump is not taken.

DUMPTYPE=F gives you a full dump of the virtual machine, as well as any saved segments it uses. This occurs on some error conditions and trace points.

DUMPTYPE=P gives you a partial dump of the database machine on certain error conditions. A dump is *not* taken when a limit error (message ARI0039E) or hardware error (message ARI0041E) occurs, or when a user specification error is detected. (If you specify the DUMP keyword in the TRACE ON command, DUMPTYPE=P also generates partial dump output to the trace file for a specified trace point in the database manager.) The partial dump provides a dump of control blocks and other dynamically obtained virtual storage in the database machine.

You can use the SET operator command to change the value of DUMPTYPE without having to stop and restart the application server. See the *DB2 Server for VSE & VM Operation* manual for more information on the SET operator command.

Note: Any changes you make using the SET command are only in effect while the application server is running. If you stop and restart the application server, it will use the settings you specified in the startup procedure.

For more information on dumps, see the *DB2 Server for VM Diagnosis Guide and Reference* manual.

EXTEND

This parameter specifies whether or not special recovery commands are processed at startup. Only set it to Y when you have a DBSS processing error or a severe user error. For more information on this parameter, see the discussion on starting the application server to recover from DBSS errors in the *DB2 Server for VM Diagnosis Guide and Reference* manual.

TRACDBSS, TRACRDS, TRACWUM, TRACDRRM, TRACDSC, TRACCONV, and TRACSTG

These parameters call the trace facilities during startup (as opposed to the TRACE operator command). Except for TRACWUM and TRACDRRM (which are not supported in single user mode), they are used primarily for tracing in single user mode, but can be set in multiple user mode if you want to start tracing as soon as possible. For information about tracing, refer to the *DB2 Server for VSE & VM Operation* manual.

TRACEBUF

This parameter specifies the amount of memory (in kilobytes) to allocate to the trace buffer. Specifying a nonzero value causes trace output to be stored in a fixed size buffer in memory. Trace records are stored in wrap-around mode in this buffer, and when tracing is turned off, the contents of the buffer are written to disk or to tape (as specified by the ARITRAC FILEDEF statement). The trace buffer is only created if you specify TRACEBUF with at least one of the startup initialization parameters TRACRDS, TRACDBSS, TRACDSC, TRACCONV, TRACDRRM, TRACWUM, or TRACSTG; it is not created if the TRACEBUF default (n=0) is specified. A suggested size for the trace buffer is 100 kilobytes or more. If you do not specify TRACEBUF and tracing is requested, trace records are written directly to disk or tape as the trace points are processed.

Single User Mode Initialization Parameters

Figure 11 on page 78 identifies the initialization parameters that apply when the database manager is operating in single user mode.

PARAMETER	DEFAULT	MINIMUM	MAXIMUM
ENVIRONMENT PARAMETERS			
DBNAME(name)	None	—	—
DCSSID()	SQLDBA	—	—
AMODE()	31	—	—
SYSMODE=S	—	—	—
STARTUP=W R	W	—	—
PARMID=name	None	—	—
CHARNAME=name	INTERNATIONAL	—	—
ACCOUNT=D N	N	—	—
PROGNAME=name	None	—	—
DSPSTATS=nn	00	00	21
PERFORMANCE PARAMETERS			
NPACKAGE=n	10	1	32766
NPACKPCT=n	30	0	100
NPAGBUF=n	14	10	400000
NDIRBUF=n	NPAGBUF	10	400000
NCSCANS=n	30	1	655
RECOVERY PARAMETERS			
LOGMODE=Y A N L I	Y	—	—
CHKINTVL=n	10	1	99999999
SLOGCUSH=n	90	11	90
ARCHPCT=n	80	10	99
SOSLEVEL=n	10	1	100
SERVICE PARAMETERS			
DUMPTYPE=P F N	F	—	—
EXTEND=Y N	N	—	—
TRACDBSS=nnn...	000...	000...	222...
TRACRDS=nnnnnnn	0000000	0000000	2222222
TRACDSC=nn	00	00	22
TRACCONV=n	0	0	2
TRACSTG=n	0	0	1
TRACEBUF=n	0	0	99999

Figure 11. Single User Mode Initialization Parameters

Most of the considerations for setting these parameters are the same as those described under “Multiple User Mode Initialization Parameters” on page 58, with the following exceptions:

- The value of SYSMODE is S, which specifies that the database manager is dedicated to a single application.
- The database manager does not generate accounting records when STARTUP=C|E|L|S|I|M, which are special situations. For more information, see the *DB2 Server for VSE & VM Operation* manual.
- The DBMODE parameter does not apply.

- The PROTOCOL parameter does not apply.
- The SYNCPT parameter does not apply.
- The PROGNAME parameter is required (except when STARTUP=C|E|L|S||M, which are special cases), to identify the application program to be run.
- The NCUSERS parameter is not used; it defaults to 1.
- The DISPBIAS parameter does not apply.
- The NLRBS and NLRBU parameters are omitted (there is no locking in single user mode).
- The LOGMODE parameter can take the value N, which specifies that changes made by the application program are not to be logged.
If LOGMODE=N, database changes are only committed when a checkpoint is explicitly taken (with COMMIT WORK statements).
The ARCHPCT parameter cannot be specified if LOGMODE=N.
- The TRACDRRM and TRACWUM parameters do not apply.

Tape Support

The database manager can use tape files for recording archive and trace information. You can also use tape files with the DATALOAD/DATAUNLOAD and RELOAD/UNLOAD facilities of the DBS utility. (It is also possible to use tape files for the DB2 Server for VM preprocessors, but this is unusual.)

For the archive and trace tape files, the IBM-supplied EXECs that starts these facilities provide default CMS FILEDEF commands for the needed tapes. These default FILEDEFs are shown under the descriptions of the EXECs that call them.

You can also take log archives to disk. For more information, see “Log Archiving to Disk” on page 219.

For the DBS utility tape files, you must supply your own CMS FILEDEF commands. You can also specify LABELDEF commands. You should use the LABELDEF command for multivolume standard label tapes.

To specify your own FILEDEF and LABELDEF commands, issue them before invoking the EXEC that calls the facility. When an IBM-supplied EXEC issues a CMS FILEDEF command for tape files, it uses the NOCHANGE parameter. This means that any FILEDEF (or LABELDEF) that you supply before running the EXEC overrides the default.

The database manager uses the CMS simulation of OS QSAM for its tape support. The database manager also provides additional support, as follows:

- Unlabeled tapes and IBM (EBCDIC) standard labels
- Multivolume tape files (with standard labels only)
- Spanned records for both input and output

The following sections discuss considerations for using tape support.

Unlabeled Tapes

When using unlabeled tape output files, you should be aware of the following:

- The mounted tape must not contain a volume label (VOL1). If it does, tape OPEN processing fails.
- For output files, if end-of-volume is reached before the tape is closed, CMS ends abnormally.
- A database or log archive cannot span multiple tapes if they are unlabeled tapes. Standard labeled tapes must be used for multiple volume tape archives.

Labeled Tapes

When using standard label tapes, you should ensure that the mounted tape volume (or volumes) contain volume labels (VOL1) and file labels (HDR1). These labels must be recorded in the same tape density as specified (or allowed to take the default) when creating the new file. If you do not ensure that the labels are recorded in the same density as specified when creating the new file, tape OPEN processing fails.

You can use the CMS TAPE command to check whether a volume contains a volume label (and display the label's contents) with the DVOL1 keyword. (You must supply the TAP n parameter as appropriate.)

You can also use the CMS TAPE command to create a volume label (VOL1) and dummy HDR1 label with the WVOL1 keyword. (Once again, you must supply the TAP n and DEN parameters as appropriate.) The tape volume label must be recorded in the same density as the file to be created. (The density of the volume label must match the CMS FILEDEF command DEN parameter value.)

You should specify LABELDEF commands for your tape files so that processes that use tapes (such as RESTORE) can verify that the correct tape is mounted. This is particularly advisable when working with multivolume tape files.

Note: If you are processing multivolume tape files, you should use a different VOLID for each tape volume so that the system can verify that the correct tape is mounted. To do this, enter VOLID ? with the LABELDEF command. CMS prompts you for the individual VOLIDs. For more information on the LABELDEF command, see the *VM/ESA: CMS Command Reference* manual for your VM system.

Single-Volume Tape Files

For single-volume tape files, you can use the following CMS FILEDEF command tape label options:

- SL if the tape has standard labels
- NL if the tape is unlabeled
- BLP if the tape has standard labels, standard user labels, or nonstandard labels with a tape mark at the end of the labels
- LABOFF if the tape is unlabeled (and has no leading tape mark).

The database manager does not support nonstandard labels or standard user labels (except with the FILEDEF BLP parameter as described in the preceding list). Therefore, you must not specify tape label options SUL or NSL in the CMS FILEDEF command.

Multiple Volume Tape Files

In addition to the FILEDEF command, you should specify a LABELDEF command for multivolume standard label tapes. This enables CMS to verify that the correct tape is mounted when a multivolume tape file is being processed.

If you have two tape drives available, you can specify an alternate tape drive in the FILEDEF command (this is only supported with labelled tapes). This causes tape drives to be switched automatically when end-of-tape is reached. If you are using a single tape drive you must mount a new tape when end-of-tape is reached.

The following is an example of FILEDEF and LABELDEF commands for a database archive:

```
TAPE WVOL1 ARCD1 (TAP1
TAPE WVOL2 ARCD2 (TAP2
LABELDEF ARIARCH VOLID ?
DMSLBD441R Enter VOLID information:
ARCD1
DMSLBD441R Enter VOLID information:
ARCD2
DMSLBD441R Enter VOLID information:

FILEDEF ARIARCH TAP1 SL 1 (ALT TAP2
```

This LABELDEF statement assumes that the archive requires two tape drives. If it requires more, you are prompted to enter more VOLIDs during the archive procedure.

Spanned Records

For spanned-record files, omit the LRECL value from the CMS FILEDEF command. If specified, it is ignored.

There are no other special considerations for spanned-record input files.

For spanned-record output files:

- If RECFM=VBS is specified and the maximum logical record size is less than the block size minus 4, the database manager changes the RECFM value to VB (in the CMS file system).
Note: Files written in RECFM=VB format can be read with the RECFM specification of either VBS or VB.
- If RECFM is not VBS, the database manager uses (in the CMS file system) RECFM U and simulates RECFM=VS. (The file is written in RECFM=VS format.)
Note: Files written in RECFM=VS format can be read with the RECFM specification of either VS or VBS.

Blocking for Archives to Tape and Disk

The block size for database archive file output and log archive file output is always 28 kilobytes.

General File Support

Many of the database manager facilities use SYSIN, SYSPRINT, and SYSPUNCH files. The IBM-supplied EXECs that call these facilities often contain parameters that allow you to assign these files to various devices. These EXEC parameters generate CMS FILEDEF commands for the files internally.

In many instances, however, the EXECs provide for only the most common files. If you want something that is not an option in an EXEC parameter, you can issue a FILEDEF command before running the EXEC. For example, to assign SYSIN to tape for the DBS utility, you must issue a CMS FILEDEF command before running the DBS utility EXEC (SQLDBSU).

Many of the usual VM assignments for SYSIN, SYSPRINT, and SYSPUNCH are valid for DB2 Server for VM use. The following list summarizes the valid assignments:

SYSIN The SYSIN files can be CMS files, virtual reader files, the virtual machine terminal, or tape and DASD SAM files supported by CMS OS QSAM. The files must contain fixed-length 80-byte logical records. Except for the virtual reader files and for terminal input, the files can be fixed block. A CMS FILEDEF command for SYSIN can specify RECFM FB and BLKSIZE *nnnn*. The *nnnn* must be some multiple of 80.

SYSPRINT The SYSPRINT files can be CMS files, virtual printer files, the virtual machine terminal, or tape SAM files supported by CMS OS QSAM. All SYSPRINT records are fixed-length, 121-byte logical records. The 1st byte is an ANSI (ASA) carriage control character. Except for the virtual printer files and terminal output, the files can be fixed blocked. A CMS FILEDEF command for a SYSPRINT file can specify RECFM FBA (or FB) and BLKSIZE *nnnn*. The *nnnn* must be some multiple of 121. If you specify RECFM in the SYSPRINT FILEDEF, you must specify FA or FBA (unless you want the ANSI carriage control characters printed). The value FA is the default.

The DBS utility and ISQL support other print file logical record sizes. In addition, ISQL supports other devices. For more specific information, refer to the *DB2 Server for VSE & VM Database Services Utility* and the *DB2 Server for VSE & VM Interactive SQL Guide and Reference* manuals.

SYSPUNCH The SYSPUNCH files (used only by the DB2 Server for VM preprocessors) can be CMS files, virtual punch files, or tape sequential files supported by CMS OS QSAM. The database manager punches fixed-length, 80-byte logical records. Except for virtual punch files, they can be fixed blocked. The CMS FILEDEF command for a SYSPUNCH file can specify RECFM FB and BLKSIZE *nnnn*. The *nnnn* must be some multiple of 80.

Remember that normal CMS defaults on FILEDEF commands apply. Specifically, if the file is a CMS file, and you do not specify a file mode, CMS uses A1. If you specify only a file mode letter, CMS uses a file mode number of 1. If you specify * for the file mode, CMS searches all accessed minidisks (A to Z) for a file with the specified file name and file type.

Starting the Application Server in Multiple User Mode

You start the application server in multiple user mode so that one or more applications can concurrently access the same application server.

To start the application server in multiple user mode:

1. Log on to a database machine
2. IPL CMS
3. Issue the SQLSTART EXEC.

Note: You cannot run the database manager in a CMS batch machine.

Figure 12 shows the format of the SQLSTART EXEC.

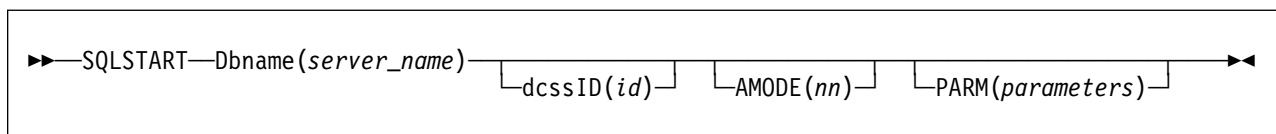


Figure 12. SQLSTART EXEC

The parameters for the SQLSTART EXEC are:

Dbname(server_name)

This parameter must be specified and must precede the PARM parameter. The *server_name* variable identifies the application server.

After initial installation and database generation, the only application server you have is named SQLDBA. If you add more databases, you can specify other names for DBNAME.

dcssID(id)

Specify this optional parameter only if you have created saved segments for the DB2 Server for VM code and want to use those saved segments, and you have generated a bootstrap package other than SQLDBA. If DCSSID is not specified, the *id* value from the *resid* SQLDBN file on the production disk is used. If DCSSID is specified, but is different from the value in the *resid* SQLDBN file, the new value is saved. If no value is available, SQLDBA is used.

If specified, DCSSID must precede the PARM parameter. You can specify ID instead of DCSSID for the keyword. No other abbreviation is valid. For more information on starting the application server to use saved segments, refer to Chapter 8, "Saved Segments" on page 181.

AMODE(nn)

This optional parameter specifies the type of addressing the database manager runs in. It has two options:

AMODE(31) When this option is specified, the database manager uses 31-bit addressing and storage above 16M can be used. This is the recommended addressing mode.

AMODE(24) When this option is specified, the database manager uses 24-bit addressing. In this case, storage above 16M cannot be used and must **NOT** be defined (ie: the virtual machine size must not exceed 16 megabytes), unless the RDS component is executed from a saved segment defined below 16 megabytes, **OR** the RDS component is linkeditted with "AMODE ANY RMODE 24."

The value specified for the AMODE parameter is saved in the *resid* SQLDBN file. If AMODE is not specified in the SQLSTART EXEC, the *resid* SQLDBN file is checked, and the AMODE value found in the *resid* SQLDBN file is used. If this file does not exist or does not contain an AMODE value, AMODE(31) is used and this value is saved in the *resid* SQLDBN file. The database manager continues to use this value until a different value is specified.

When AMODE is specified in the SQLSTART EXEC, this parameter must precede the PARM parameter. No abbreviation of AMODE is valid.

Single user mode applications and user exits will be invoked in the same addressing mode as the database manager. If you have such applications that do not support 31-bit addressing, you must do one of the following:

- Convert your application programs so you can exploit 31-bit addressing
- Use the AMODE(24) option of the SQLSTART EXEC.

For more information on converting your program, see the *VM/ESA: CMS Application Migration Guide*. For more information on single user mode, see "Starting the Application Server in Single User Mode" on page 86. For more information on user exits, see Chapter 14, "Creating Installation Exits" on page 355.

Note that the preprocessors and the DBS utility must run in 24-bit addressing mode. In single user mode, if the database manager is running AMODE(31), the AMODE is automatically switched to AMODE(24) before invoking the preprocessor or DBS utility. The AMODE is then switched back to AMODE(31) after control is returned to the database manager.

The resource adapter always runs AMODE(31) in XA mode or XC mode regardless of the mode the database manager is running in.

PARM(parameters)

This optional parameter is used to specify initialization parameters and user application program parameters. If specified, it must be placed last, after DBNAME, DCSSID, or AMODE. When specifying initialization parameters, separate them with a comma or a blank.

Note: For users moving from the database manager on VSE to the database manager on VM. The same parameters that are supported on VSE are supported on VM. The exceptions are the DSPLYDEV and DBPSWD parameters.

The database manager on VM ignores the DSPLYDEV parameter. Instead, SQLSTART always issues SP CON START HOLD (unless the database manager is already spooled START), and all output (except dumps) goes to the console. Dumps go to the virtual printer or reader. This implementation is different because, on the VSE operating system, there is only one operator console and one SYSLST for each partition. In VM, all machines usually have their own console and virtual printer.

The DBPSWD parameter was used in VSE to specify a VSAM password. This parameter does not apply to the database manager on VM, and is ignored if specified.

During its processing, SQLSTART issues these CMS FILEDEF commands for the trace and archive files:

```
FILEDEF ARIARCH TAP1 SL (NOCHANGE PERM
FILEDEF ARITRAC TAP2 SL (NOCHANGE PERM
FILEDEF ARILARC TAP3 SL (NOCHANGE PERM
```

To override these FILEDEF commands, issue your own before running SQLSTART. You must use the ddnames ARITRAC, ARIARCH, and ARILARC for the trace, database archive, and log archive files, respectively. Standard label, unlabeled, single volume, and multivolume tapes are supported. For more information on tape support, see “Tape Support” on page 79.

If you are using standard label tape files for tracing, database archiving, or log archiving, you can optionally submit CMS LABELDEF commands before running the SQLSTART EXEC. This allows you to specify values to be used for file header label checking and creation. You should supply CMS LABELDEF commands to ensure that you have the proper tape files and volumes mounted. You must use the LABELDEF command for multivolume standard label tapes. For more information, see the *VM/ESA: CMS Command Reference* manual.

Do not specify any VOLID parameter on your LABELDEF (or FILEDEF) commands for log archiving (ddname ARILARC). Because more than one log archive file can be read or created during one database-manager session, you should use different VOLIDs for the different files.

You can take log archives to disk rather than tape by changing the FILEDEF of ARILARC. For more information on directing log archives to disk, see “Log Archiving to Disk” on page 219.

It is possible to direct the trace output to a memory buffer or to a CMS file rather than to a tape. This may be convenient if you often use the security audit trace. For more about directing trace output to a memory buffer or to a CMS file, see the *DB2 Server for VSE & VM Operation* manual.

Running Multiple User Mode Application Programs

When the application server is started in multiple user mode, and the user machine is initialized (with the SQLINIT EXEC), SQL application programs can be started by normal means (such as the CMS LOAD or START commands).

For more information on running application programs, see the *DB2 Server for VM Application Programming* manual.

Note: If you plan to run your application programs in both multiple user mode and single user mode, you should follow the protocols discussed in the section “CALL/RETURN Protocols for Application Programs in Single User Mode” on page 91.

Starting the Application Server in Single User Mode

An application program running in single user mode runs in the same machine as the application server, and is under its control. (In this situation, the user machine and the database machine are actually the same machine.) To run a single user mode application program, start the application server in single user mode (SYSMODE=S) and provide the program name as an initialization parameter (PROGNAME=*name*). For PROGNAME specify the name you would specify if running the program in multiple user mode. The program is loaded and control is passed to it after the application server is started. For single user mode, only the TEXT files need to be available. If you choose this method, you should put the files in a TXTLIB, because the database manager does not issue INCLUDE commands. It is preferable to create a module using the CMS LOAD/GENMOD commands, especially if the program is to be used frequently.

Your application is invoked in the same addressing mode as the database machine. If your single user mode application program does not support 31-bit addressing, you must do one of the following:

- Convert your application programs so you can exploit 31-bit addressing,
- Use the AMODE(24) option of the SQLSTART EXEC. See the RDS restriction when using AMODE(24), “Starting the Application Server in Multiple User Mode” on page 83.

For more information on converting your program, see the *VM/ESA: CMS Application Migration Guide*

Attention: The value specified for the AMODE parameter is saved in the *resid* SQLDBN file between invocations of the SQLSTART EXEC. If AMODE is not specified, the *resid* SQLDBN file is checked, and the last value is used. If you only want AMODE(24) for single user mode applications, be sure to specify AMODE(31) when restarting in multiple user mode. When running AMODE(24) option, you cannot use any storage above 16M.

Some administrative tasks (such as adding dbextents and adding dbspaces) are performed by running IBM-supplied EXECs in single user mode. These EXECs call the SQLSTART command with the appropriate parameters.

Figure 13 shows how to run an application program in single user mode. When the application server is started, it passes control to the application program specified by the PROGNAME parameter. All other initialization parameters are allowed to default. You may want to specify some single user mode initialization parameters. For information on single user mode initialization parameters, see Figure 11 on page 78.

Note: The PROGNAME parameter is not needed when STARTUP=C|E|L|S|I|M is specified. These startups specify the operation to be performed, so a program name is not needed. Moreover, the database manager provides separate EXECs for each of these situations, one of which must be used instead of SQLSTART. (Each of these EXECs calls SQLSTART at the proper time.)

```
SQLSTART DB(SQLDBA) PARM(SYSMODE=S,PROGNAME=name)
```

Figure 13. Starting in Single User Mode

During its processing, SQLSTART issues these CMS FILEDEF commands for the trace, database archive, and log archive files:

```
FILEDEF ARIARCH TAP1 SL (NOCHANGE PERM
FILEDEF ARITRAC TAP2 SL (NOCHANGE PERM
FILEDEF ARILARC TAP3 SL (NOCHANGE PERM
```

To override these FILEDEF commands, issue your own before running SQLSTART. You must use the ddnames ARITRAC, ARIARCH, and ARILARC for the trace, database archives, and log archives, respectively. Specify the PERM option on your FILEDEF commands if the application program is written in a language other than Assembler. Standard label, unlabeled, single volume, and multivolume tapes are supported. For more information on tape support, see "Tape Support" on page 79.

If you are using standard label tape files for tracing, database archiving, or log archiving, you can optionally submit CMS LABELDEF commands before running the SQLSTART EXEC. This allows you to specify values to be used for file header label checking and creation. You should supply CMS LABELDEF commands to ensure that you have the proper tape files and volumes mounted. You must use the LABELDEF command for multivolume standard label tapes. For more information, see the *VM/ESA: CMS Command Reference* manual.

You should not specify VOLID parameters on any LABELDEF or FILEDEF commands you issue for the log archive files. Because more than one tape file can be created during a database manager run, you should use different VOLIDs for the different tape files.

It is possible to direct the trace output to a CMS file rather than to a tape. This may be convenient if you often use the security audit trace. For more information about tracing, see the *DB2 Server for VSE & VM Operation* manual.

In addition to the FILEDEFs for archiving and tracing, SQLSTART issues the following FILEDEF command for a user LOADLIB:

```
FILEDEF ARIUSRDD DISK USERLIB LOADLIB * (NOCHANGE
```

If you want to specify a different LOADLIB, issue your own FILEDEF command before calling SQLSTART. You must use the ddname ARIUSRDD on the FILEDEF command to identify a user LOADLIB to the database manager. Note that the file mode used in the FILEDEF is *. Remember to access the minidisk containing your LOADLIB ahead of other minidisks that contain USERLIB LOADLIBs.

If you use AMODE(24), the application program must be RMODE 24. If it is not, and the program is loaded above 16M, the database manager issues message ARI0021E.

The application program can be a module, or it can reside in a LOADLIB (or, conceivably, a saved segment). Because the database manager does not know where the user program is (and there are many ways to load a program in VM), the database manager tries a sequence of VM commands to load the program. The following sequence is used:

1. CMS LOADMOD command: for CMS files with a file type MODULE. The program is loaded into the CMS user program area.
2. CMS NUCXLOAD command: for members of CMS or OS LOADLIBs. The program is loaded into free storage.

3. CP diagnose FINDSYS/LOADSYS instructions: for saved segments.
4. CMS LOAD command: for TEXT files or TXTLIB members. The program is loaded into the CMS user program area. Note that CMS INCLUDE commands are not issued in this situation. Also, a GLOBAL TXTLIB command must be issued before SQLSTART if the text files are TXTLIB members. More than one library can be specified on the GLOBAL command.

Because not all of the above VM load functions return the entry point, you should code your program so the entry point is the same as the load point. Only LOADMOD and NUCXLOAD return the program's entry point. When the database manager finds a program with the name specified in the PROGNAME initialization parameter and successfully loads the program, the search sequence ends. Control is passed to the program with a BALR instruction.

For example, suppose you have two programs on your A-disk. One is named MYPROG MODULE A; the other is named MYPROG TEXT A. If you run SQLSTART with PROGNAME=MYPROG, the program loaded is MYPROG MODULE A. The database manager tries the LOADMOD command before the LOAD command. If you want to load MYPROG TEXT A, you must either rename it and change the PROGNAME parameter value accordingly, or you must rename (or erase) MYPROG MODULE A.

If the database manager does the entire search sequence, and a *not found* indication is received from each of the load functions, four messages are issued:

- ARI0026E** Indicates an error occurred while attempting to load the program.
- ARI0039E** Indicates a limit error occurred.
- ARI0042I** Indicates the reason code is 4.
- ARI0043I** Indicates the return code is 512.

If the database manager receives an insufficient storage indication from any of the load functions, the same four messages are issued, but the reason code in message ARI0042I is 8 (not 4).

If the load of the user program fails for a reason other than those discussed above, the database manager issues message ARI0026E. Following that message is one or more occurrences of message ARI0047E. Message ARI0047E has the format:

xxxxxxx- Reason Code=*nnn*

The type of load is indicated in xxxxxxxx. The xxxxxxxx can be LOADMOD, NUCXLOAD, SEGMENT LOAD, FETCH, or LOAD. The system return code from that load is in *nnn*. This message is followed by either message ARI0039E or ARI0040E, depending on the type of error. Following that message is ARI0042I with a reason code of 0 (the reason code is given earlier in message ARI0047E), and finally by message ARI0043I with a return code of 516.

Note that, for a NUCXLOAD or SEGMENT LOAD, the database manager must do more than issue those commands.

For NUCXLOAD, the sequence of NUCEXT QUERY, NUCXDROP, NUCXLOAD, and NUCEXT QUERY may be processed to load the code. Thus, the reason code displayed in message ARI0047E can be a return code from NUCEXT QUERY or NUCXDROP. (The return code is not necessarily from a NUCXLOAD.) You should

check the *DB2 Server for VM Messages and Codes* manual for return codes from NUCEXT QUERY and NUCXDROP as well as NUCXLOAD.

The database manager follows this process when attempting to load a program with NUCXLOAD:

1. Issues NUCEXT QUERY to see if a copy of the code already exists in storage (storage is not properly reset). If so, the CMS NUCXDROP command is issued. If the NUCXDROP return code is not zero, the return code is displayed as the ARI0047E reason code.
2. If the NUCEXT QUERY is successful (and NUCXDROP, if performed), the NUCXLOAD is issued. If the load fails, the nonzero return code becomes the reason code in message ARI0047E.
3. If the load is successful, another NUCEXT QUERY is issued to obtain the code load address and the code length. If this fails, a reason code of 253 is displayed in message ARI0047E.

A similar process is done for code that is to be loaded into a saved segment. The database manager does a SEGMENT FIND instruction to get the code load address and length. The SEGMENT LOAD instruction is then issued. Thus, the reason code displayed in message ARI0047E can be a return code from the SEGMENT FIND (not necessarily the SEGMENT LOAD) instruction. You should check the *DB2 Server for VM Messages and Codes* manual for return codes from SEGMENT FIND as well as SEGMENT LOAD.

The database manager follows this procedure when attempting to load a program with the SEGMENT LOAD macro:

1. Issues a SEGMENT FIND instruction to get the load address and length of the code to be loaded.
2. If the SEGMENT FIND condition code is 2, an error occurred. The return code XXX is displayed as the reason code in message ARI0047E:

```
ARI0047E  SEGMENT LOAD  - Reason Code=XXX
```
3. If the condition code is 1 (saved segment not yet loaded), the database manager does some checking before attempting to load the code: If it is not, the reason code 400 is displayed in message ARI0047E.
 - a. If the above check was successful, the database manager then checks to ensure that loading the code at the indicated load address does not overlay other database manager code. If an overlay would result, the database manager displays a reason code of 500 in message ARI0047E.
 - b. If both checks are successful, the SEGMENT LOAD instruction is issued. The code is loaded at the address returned by the SEGMENT FIND instruction.
4. If the SEGMENT LOAD is issued and the condition code is 2, an error occurred. The SEGMENT LOAD return code is displayed as the reason code in message ARI0047E.
5. If the SEGMENT LOAD is issued and the condition code is 1, a code overlay occurred. A reason code of 500 is displayed in message ARI0047E.

In addition to the loading sequence, you should be aware of the following when preparing to run a single user mode program:

- If the program resides in an OS LOADLIB, you must ensure that the proper GLOBAL and FILEDEF commands are issued before starting the application server.
- The database manager uses CMS OS QSAM for sequential file support. The CMS OS QSAM support uses the GETMAIN area of the virtual machine. The CMS OS QSAM support is called before the user's application program. You should not issue the CMS STRINIT macro in the application program, as this may release all GETMAIN storage currently allocated by the database manager. This can only occur if the setting of the CMS STORECLR option is 'ENDCMD'. When the STORECLR option is set to 'ENDSVC' (the CMS default), the STRINIT macro is ignored.
- If the application does not support 31-bit addressing, you must use AMODE(24).
- When running AMODE(24), single user mode applications (and user exits) should not switch to AMODE(31) and branch to other applications above 16M unless: those applications have no interaction or interface with DB2 Server for VM code, and, AMODE(24) is reset before returning control to the database manager.

Specifying User Parameters

If you start the application server in single user mode, you can also specify user parameters to be passed to the application program using the PARM keyword of the SQLSTART EXEC. The SQLSTART EXEC purges the CMS program and console stacks. Thus, any program run in single user mode cannot rely on console or program stack input.

Place a slash (/) between the database manager initialization parameters and the user parameters, as shown in Figure 14.

```
SQLSTART DB(SQLDBA) PARM(SYSMODE=S,PROGNAME=PROG1/parm1,parm2)
```

Figure 14. Starting in Single User Mode and Providing User Parameters

Note: Only the first 130 characters of the command line are read by CMS. The exception to this rule occurs when SQLSTART is called from a user-written EXEC; then CMS reads the first 256 characters. If you specify many initialization parameters and user parameters, they will not fit on the command line. Thus, you must use a CMS file for some of the parameters. Because user parameters cannot be specified in a CMS file, you should specify the initialization parameters in the CMS file, and the user parameters on the command line.

The user parameters are passed to the application program with register 0. Register 0 points to an area called NPLIST, which contains three addresses, which point to:

1. COMVERB, the command name, the name of the application program specified in the PROGNAME initialization parameter.
2. BEGARGS, the start of the user parameter string.
3. ENDARGS, the byte following the last character of the user parameters.

The user parameter string is untokenized: it has not been separated into individual user parameters. This pointer scheme is similar to the one that the EXEC 2 interpreter uses when running programs. Figure 15 on page 91 shows how register 0 points to the user parameters.

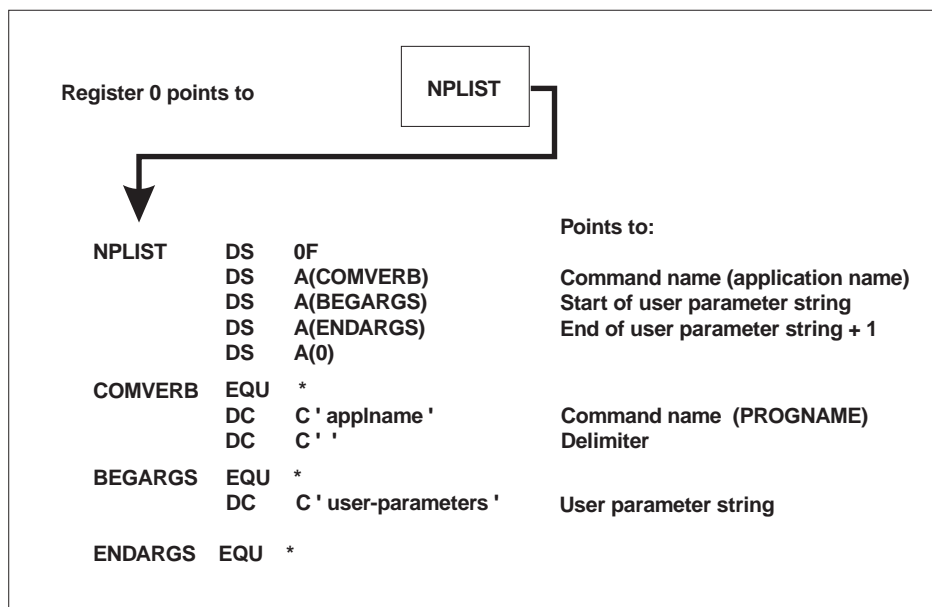


Figure 15. Passing User Parameters to a User Application Program

The length of the parameters can be obtained by subtracting the BEGARGS address from the ENDARGS address. If there are no user parameters, the ENDARGS address is equal to the BEGARGS address (ENDARGS - BEGARGS = 0.) Both addresses, in this situation, point to the next byte after the name of the application.

User parameters are not displayed along with the initialization parameters. User parameters cannot be specified in a DB2 Server for VM parameter data set.

CALL/RETURN Protocols for Application Programs in Single User Mode

In single user mode, an application is called using normal CALL/RETURN protocols, as follows:

- Register 0** Pointer to pointer to user parameters
- Register 1** Contains zeros
- Register 13** Pointer to DB2 Server for VM save area
- Register 14** Return point to the database manager
- Register 15** Entry point of the user program.

Note: This same protocol can also be used by programs running in multiple user mode.

Register 0 was discussed in the previous section. A program written in C, PL/I, COBOL, or FORTRAN requires an interface routine to process the user parameters.

Upon entry, the application program must store the registers in the DB2 Server for VM save area, and restore them before returning control to the database manager. Failure to do so causes unpredictable results.

An abnormal termination exit is set to intercept abnormal termination conditions, including program checks. If the user program establishes its own abnormal end exit, the user exit overrides the DB2 Server for VM abnormal end exit.

The abnormal end exit is set (with CMS ABNEXIT) to intercept abnormal end conditions (including program checks). If the user program establishes an abnormal end exit (for example, with ABNEXIT, STAE, SPIE), the user program gets control before the DB2 Server for VM abnormal end exit does. (However, STAE and SPIE are not supported in 31-bit addressing mode.) When the user program completes its abnormal end processing, it should return control to the CMS abnormal end routine. The CMS abnormal end routine then passes control to the DB2 Server for VM abnormal end routine. The application programmer must be careful when processing abnormal end conditions. These possibilities must be considered:

1. The abnormal end condition occurs in your program, and you can determine that this is the case. You can then circumvent the problem and continue processing. In this case, reset the abnormal end exit. Otherwise, future abnormal end conditions cause control to be given to the DB2 Server for VM abnormal end routine.

If you determine that processing cannot continue, you should reset your abnormal end exit, clear the abnormal end exit, and return control to the CMS abnormal end routine. The CMS routine then passes control to the DB2 Server for VM abnormal end routine.

2. If you cannot determine anything about the abnormal end condition, you should reset the abnormal end exit, clear the abnormal end exit, and return control to the CMS abnormal end routine. (You would do so when, for example, you did not know if your program caused the abnormal end.) The CMS abnormal end routine then passes control to the DB2 Server for VM abnormal end routine.
3. Finally, consider the situation when no abnormal end condition occurs, and your program ends normally (control is returned to the database manager). In this situation, your abnormal end exit should be cleared prior to returning control to CMS. When both DB2 Server for VM processing and user processing finish, both abnormal end exits must be cleared. Otherwise, future abnormal end conditions in the virtual machine could cause unpredictable results.

The DB2 Server for VM abnormal end routines sever the IUCV links to the database minidisks, and close the trace file if tracing was activated. This same processing is also done when, upon completion, the user program returns control to the database manager. The database manager does not have to do this processing (for example, if the program does not return control to the database manager). If the IUCV links to the database minidisks are not severed, VM severs the links when you log off the database machine. (This also is true if the database machine abnormally ends.) If tracing or accounting were active, their output files may not have had the last buffers written. If the output files were on tape, no tape mark was written. You can then write tape marks with the CMS TAPE command.

The database manager uses eye-catcher technique for determining when a specific module is in error. The eye-catcher is displayed in the DB2 Server for VM mini-dump. A user program can use the same technique in single user mode,

assuming that the DB2 Server for VM abnormal termination exit has not been overridden by a user abnormal end exit. A suggested coding example in assembler language is shown in Figure 16 on page 93.

```

        USING  *,15
        B      SKIPEYE      BRANCH AROUND EYE-CATCHER
        DC     AL1(16)      LENGTH OF CHARACTER STRING
        DC     CL8'progname' PROGRAM NAME EYE-CATCHER
        DC     CL8'&SYSDATE' DATE PROGRAM COMPILED
        DS     0H
SKIPEYE EQU  *
        STM    14,12,12(13) SAVE DB2 Server for VM  REGISTERS
        BALR   12,0         ESTABLISH BASE REGISTERS
        DROP   15
        USING  *,12
        LA     11,MYSAVEAR  GET ADDRESSABILITY TO MY SAVE AREA
        ST     11,8(13)     SAVE ADDRESS OF SAVE AREA IN DB2 Server for VM  SAVE AREA
        ST     13,MYSAVEAR+4 SAVE ADDRESS OF DB2 Server for VM  SAVE AREA IN SAVE AREA
        LR     13,11       SET REGISTER 13 TO MY SAVE AREA
        .
        Body of the Application Program
        .
EXIT    L      15,RETCOD    SET RETURN CODE (OR SET TO ZERO)
        L      13,4(13)    GET DB2 Server for VM  SAVE AREA
        L      14,12(13)   GET DB2 Server for VM  REGISTER 14
        LM     0,12,20(13) GET OTHER DB2 Server for VM  REGISTERS
        BR     14         RETURN TO DATABASE MANAGER

```

Figure 16. Use of an Eye-catcher by an Application Program

Notes:

1. The instruction BALR 15,0 can be used just ahead of the USING *,15 instruction as long as other registers are not used until the DB2 Server for VM registers have been saved.
2. The techniques shown here work whether the application program is called by the database manager, or is called as a CMS command. Thus, the same application program can be run in either single or multiple user mode.
3. The techniques shown here may not always be achievable by a FORTRAN, C, COBOL, or PL/I program. A program written in one of these languages may need to be called by a pre-entry routine, to ensure that register 15 contains a zero (or valid return code) upon return to the database manager.

Overriding Initialization Parameters

When starting the application server, you can change the default parameter values in either of two ways:

- By specifying the parameters in the PARM field of the SQLSTART EXEC.
- By creating a CMS file that contains DB2 Server for VM parameters and calling it with the PARMID initialization parameter. See Figure 9 on page 60 for an example.

You can also combine the two methods. Parameters specified in the CMS file override the default values. Parameters specified on the SQLSTART EXEC

override both the default values and those specified in the CMS file. A user who has a CMS file with an incorrect parameter value can override the value in error with a correct specification on the SQLSTART EXEC.

When all the values of the initialization parameters have been resolved, the final values (or defaults, if no values have been overridden) are displayed on the DB2 Server for VM operator's console.

When you specify parameters on the SQLSTART EXEC, separate each parameter with a comma or blank. For example:

```
SQLSTART DBNAME(SQLDBA) PARM(DUMPTYPE=F,LOGMODE=A)
```

```
SQLSTART DBNAME(SQLDBA) PARM(DUMPTYPE=F LOGMODE=A)
```

Because CMS reads only the first 130 positions of the CMS command line, you may choose to set up your initialization parameters in one or more CMS files. Such an arrangement allows you to specify more user parameters (if any) when running application programs in single user mode. User parameters (those for the application program itself), cannot be specified in a CMS file, and must be specified in the PARM field of the SQLSTART EXEC. If you plan to use user parameters, refer to "Specifying User Parameters" on page 90.

You can also call the SQLSTART EXEC from within a user-written EXEC.

Creating a Parameter File

You can store various parameters in a CMS file that has a file type of SQLPARM, and a fixed record length of 80 bytes. To have the database manager use the file, specify the file name in the PARMID initialization parameter. Each file can start the application server for a slightly different environment. Figure 17 shows a parameter file.

```
LOGMODE=A,NDIRBUF=20,  
NPAGBUF=20,  
DUMPTYPE=F                                COMMENT -- FULL VIRTUAL MACHINE DUMP  
NCSCANS=20
```

Figure 17. Example of an Initialization Parameter File

The rules for specifying parameters in a CMS file are a little different from the rules for specifying parameters on the SQLSTART EXEC:

- The parameters must be in uppercase in a parameter file.
- Because a blank after a parameter ends the processing of the line, do not put a blank between parameters. Anything on the line after that blank is ignored. You can, however, use this arrangement to put comments in the file, as shown in Figure 17 for the DUMPTYPE parameter.
- A comma at the end of a line is not required, but can be used to make the statement easier for you to read.
- User parameters (those destined for the application program itself) cannot be specified in a parameter file. If the database manager detects parameters other than its own initialization parameters, it issues error messages and stops.

Running the Database Manager

When you use the database manager, you should be aware of the following:

- Saved segments are defined using VMSES/E.
- The resource adapter and the RDS component can be saved above the 16 megabyte line.

For more information, see “Defining Saved Segments” on page 187.

- Operating modes

Your virtual machine can be set to XA/ESA mode or XC mode. If certain components are defined as saved segments, they must be saved below the 16 megabyte virtual storage line. (The exceptions are the resource adapter and RDS, which can be saved above the 16 megabyte line.) The AMODE parameter specifies the type of addressing the database manager runs in. For more information, see “Starting the Application Server in Multiple User Mode” on page 83.

A user machine can run in XA/ESA mode or XC mode. It can take advantage of a resource adapter saved segment saved above the 16 megabyte virtual storage line.

User application programs can take advantage of 31- or 24-bit addressing, and reside above or below the 16 megabyte virtual storage line.

For more information on running application programs in either XA/ESA or XC mode, see the *DB2 Server for VM Application Programming* manual.

- The following facilities are available:
 - VSE guest sharing
 - CMS work unit support
 - Database switching
 - Remote unit of work
 - Distributed unit of work.

Operating Modes

You set the virtual machine operating mode in the machine’s CMS directory, or with the SET MACHINE command. Users can issue this command for their virtual machines, and the operator issues it for the database machine. For more information on this command, see the *VM/ESA: CMS Command Reference* manual.

Table 7 on page 96 shows how features can take advantage of the VM/ESA ESA Feature.

<i>Table 7. Summary of Support</i>				
	AMODE		RMODE	
	24	31	24	ANY
RA/DRRM/CONV	X	X	X	X ¹
DSC	X	X ²	X	
RDS/DRRM/WUM/CONV	X	X	X	X
DBSS	X	X ²	X	
ISQL	X		X	
DBSU	X ³		X	
Preprocessors	X ³		X	
User Applications	X	X ⁴	X	X ⁴

Notes:

1. The resource adapter (RA) runs AMODE(31) RMODE(ANY). It does not depend on the AMODE parameter.
2. This DB2 Server for VM code must reside below the 16 megabyte line. However, most dynamic storage is allocated above it (if available).
3. In single user mode, if the database manager is running AMODE(31), it automatically switches to AMODE(24) when the preprocessor or DBSU is invoked by the database manager. The AMODE is then switched back to AMODE(31) after returning control to the database manager.
4. For more information, refer to “Starting the Application Server in Multiple User Mode” on page 83, “Starting the Application Server in Single User Mode” on page 86, and Chapter 14, “Creating Installation Exits” on page 355.

Disconnecting the Database Machine

You can free up the database machine console in two ways:

- Stop the application server and log off the database machine
- Disconnect from the database machine (and leave the database manager running).

To log off the database machine, stop the application server by using the `SQLEND` command, and then log off. Stopping the application server is explained in “Stopping the Application Server” on page 97. If you want to sign off the database machine, and leave the database manager running, enter these commands:

```
#CP SET RUN ON
#CP DISCONN
```

You should not leave the operator console unattended. To protect the integrity of your database, always have the operator sign off the operator console with the `DISCONN` command before leaving the console.

Stopping the Application Server

This section discusses the following topics:

- Taking an archive
- Verifying the directory
- Online support considerations for VSE guest sharing
- Minidisk passwords
- Inter-machine communications

In single user mode, the application server stops itself when the task is completed. In multiple user mode, the operator stops it by issuing the SQLEND operator command. In both modes, the database files and the trace file (if active) are closed. The SQLEND command is described in the *DB2 Server for VSE & VM Operation* manual.

The SQLEND command can be entered from the operator console of a database machine. Its format is shown in Figure 18. The ARCHIVE, LARCHIVE, and UARCHIVE parameters are used to initiate archive activities after the database has been shut down, and are discussed in the next section. The NORMAL parameter is used to shut down the database when all work in progress is completed. The QUICK parameter is used to stop all work in progress and shut down immediately. The TRCPURGE parameter is used if you want to purge the contents of the trace buffer at DB2 Server for VM shut down. You can also specify the DVERIFY parameter to do a directory verification.

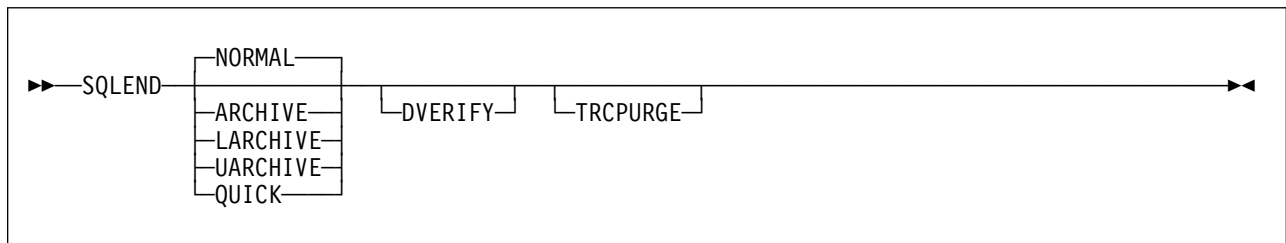


Figure 18. SQLEND Operator Command

Do not issue SHUTDOWN from the VM console as it shuts down VM and causes the database manager to end abnormally.

Taking an Archive

The SQLEND command can be set up to enable the operator to take a database or log archive after all DB2 Server for VM activity has stopped. The following parameters are available for archiving:

- ARCHIVE for a database archive using DB2 Server for VM facilities
- LARCHIVE for a log archive using DB2 Server for VM facilities
- UARCHIVE for a database archive using user facilities.

Attention: User archive facilities are available for the database, but not the log. Never attempt to use user facilities to archive a log.

The most appropriate time to take an archive is at shutdown, so consider setting up a procedure for periodic SLENDS with the ARCHIVE, UARCHIVE, or LARCHIVE parameters, as needed.

For both database and log archives, online archives are disruptive to users. Taking archives during SLENDS avoids this disruption. In addition, database archives taken at SLENDS contain data that is consistent, whereas those started by operator ARCHIVE commands or triggered by ARCHPCT typically contain uncommitted or incomplete data, and require information from the log to make the data consistent. (Consistency is not a problem for log archives regardless of when they are taken, because the database manager always waits until all LUWs end before taking the checkpoint on which the log archive is based.)

To determine the best recovery procedures for your installation, see “Recovering from DASD Failures that Damage the Database” on page 207.

If the operator specifies ARCHIVE or UARCHIVE when LOGMODE=Y, the database manager automatically switches the LOGMODE to A. To resume running with LOGMODE=Y, the operator must do a COLDLOG. See “Switching Log Modes” on page 237.

Should you decide *not* to take an archive at shutdown, specify NORMAL or QUICK. During a normal shutdown, the database manager allows all active LUWs to finish before ending. During a quick shutdown, the application server ends immediately: in-progress LUWs receive a negative SQLCODE and are rolled back the next time the application server is started.

Note: A User Archive will **NOT** be consistent if it is taken following an SLENDS QUICK shutdown.

If you are running with LOGMODE=L, and request a database archive, and if there is data in the log, then the database manager takes a log archive before taking the database archive. This log archive is written to tape. However, you can direct it to disk if you change the FILEDEF for the log archive file, or if you direct the log archive to disk when you archive it. For more information on directing log archives to disk, see “Log Archiving to Disk” on page 219.

Database archives are written to tape. When running a database archive, the database manager displays external label information for you to write on the tape if you are archiving to tape. It then requests that you mount the required volumes. If you are archiving to disk, you should respond by typing the virtual device address. Unless you have issued your own CMS FILEDEF command before starting the application server, the virtual device address for database archives is 181. The virtual device address for log archives (either explicitly requested or automatically created) is 183. See “Archiving Procedures” on page 211 for more information.

When the SLENDS command is issued with the NORMAL, ARCHIVE, LARCHIVE, or UARCHIVE parameters, a shutdown is not initiated until all users are disconnected from the application server. The database manager displays a message showing how many agents are still active. (An agent is an internal representation for a user.) As each agent becomes inactive, another message is displayed with an updated count.

The initial count displayed in the message includes all *active* user agents. When users who are inactive (not allocated to a real agent) disconnect from the database

manager, no message is displayed to indicate a reduction in agents; the message is issued only when a user disconnects from the database manager while still allocated a real agent. This results in gaps in the updated count messages.

After issuing an SQLEND command, and before shutdown commences, the operator can issue a SHOW ACTIVE command to find out who is still using the database manager. Users who are connected with no active LUW can prevent the database manager from performing shutdown operations. For example, an ISQL user can end an LUW and then leave the terminal without exiting from ISQL. To determine whether inactive users are preventing the shutdown operation, use the SHOW USERS operator command to determine which users are still active. For more information on the SHOW commands, see the *DB2 Server for VSE & VM Operation* manual.

If the SQLEND command is issued with the QUICK parameter, all in-progress work ends and return code 508 is displayed on the console. This command can be issued at any time, even following an SQLEND issued with another parameter.

Verifying the Directory

The DVERIFY parameter determines whether the database manager checks for inconsistencies in the directory. It can be specified with the other parameters, but is ignored if you specify QUICK. It should be specified each time the database is archived (using either DB2 Server for VM or user facilities); if it is not, any inconsistency in the directory will be recorded in the database archive, so a subsequent restore operation using that archive would fail.

Even if you have not requested a database archive, you should periodically verify the directory (perhaps every few days, depending on the volume of update activity). Otherwise, inconsistencies may surface later. For example, an inconsistency can cause an abnormal end during checkpoint processing. Early detection reduces data loss.

If an error is found in the directory, a message is displayed. If this happens, and you had specified ARCHIVE, the archive is not taken. If you had specified UARCHIVE (a database archive using user facilities), then when you are prompted to take the archive, do not do so. However, if you had specified LARCHIVE, the log archive *is* taken; the inconsistency in the directory does not affect the log, so the log archive is still valid. For information on recovering from directory verification errors, see the *DB2 Server for VM Diagnosis Guide and Reference* manual.

Online Support Considerations for VSE Guest Sharing

If you are supporting an online (CICS) environment, you should stop the online support before ending the application server, in order to clean up CICS transaction processing efficiently. To stop the online support, enter the CIRR or CIRT transaction. For more information on the effect of a shutdown on online applications, see “Stopping the Online Support -- The CIRT Transaction” on page 135 and “Removing Connections -- The CIRR Transaction” on page 123.

Note: For DB2 Server for VSE, each link from the Online Support requires a dedicated agent, whether or not these agents are actually active. SQLEND NORMAL will not terminate these connections.

A Note about Minidisk Passwords

Many of the IBM-supplied EXECs described in this chapter (and throughout the manual) access the DB2 Server for VM production and service minidisks. These EXECs often must write to and read from those minidisks.

Depending on the tasks you are trying to do and the virtual machine you are using, you can be prompted for the read, write, or multiple access passwords for the minidisks.

You should always be prepared to supply the passwords for the production and service minidisks before you run the IBM-supplied EXECs.

Note: DB2 Server for VM users should not know the passwords for the production and service minidisks, or any other database machine minidisks.

Inter-Machine Communications

Advanced Program-to-Program Communication/Virtual Machine (APPC/VM) is used by software to communicate between user and database machines, regardless of their physical locations. The Inter-User Communication Vehicle (IUCV) is limited to communications between two virtual machines residing on the same processor.

Internally, the database manager uses NCUSERS to determine the number of agent structures to create. Each agent structure serves one user at a given time. (That is, one user who is within an LUW.) Processing time is divided among the agent structures. You can think of an agent structure as equivalent to a user for whom the database manager is currently doing work. Thus, NCUSERS controls the number of concurrent users (agent structures) using the database manager.

As discussed earlier, each agent structure uses virtual storage and produces some processor overhead. If NCUSERS is set too high for your particular system configuration, the database manager may become overloaded and perform poorly. To determine the optimal NCUSERS setting for your installation, use the guidelines given in "NCUSERS" on page 66.

The optimum number for NCUSERS is usually less than the total number of users planned for a database. Thus, the number of connected users trying to access a database machine usually far exceeds the number specified for NCUSERS. For example, if there are 80 users and only 8 agent structures, all 80 users would be competing for those structures.

To solve this problem, the number of connected users can exceed the number specified for NCUSERS. The number of users that can be connected is related to a value called MAXCONN.

The MAXCONN parameter of the VM OPTION directory control statement determines the maximum number of IUCV connections allowed for a virtual machine. For inter-machine communications, the virtual machine is the database machine. MAXCONN has a default value of 16.

The database manager uses APPC/VM (or IUCV) to access the database minidisks (including the directory, the logs, and the dbextents) and to communicate with user machines. Thus, the number of users that can be connected to a database machine is equal to the value of MAXCONN minus the number of minidisks for the database currently being accessed. On a VM/ESA operating system, the number of

users that can be connected is decreased by one more because the DB2 Server for VM machine makes an additional connection to CP system service *IDENT. It is further reduced by one if the special TCP/IP communications real agent is active.

Usually, MAXCONN is set when a database machine is defined. (For more information, see “Adding a Primary Database Machine” on page 287.) This initial setting is based on an estimate of the number of minidisks that make up the database and the number of users. As these conditions change, MAXCONN should be readjusted.

Because the number of connected users can exceed the number of real agents, the database manager uses another mechanism to keep track of users that are not assigned to real agents. This mechanism is called the *pseudo-agent structure*.

The number of pseudo-agents is equal to the value of MAXCONN minus the number of minidisks for the database currently being accessed. Initially, these pseudo-agents are placed on an available queue. When an APPC/VM (or IUCV) CONNECT to the database manager is issued, the user is assigned to a pseudo-agent, and placed in an in-use queue. When a user issues a statement (for example, SELECT), the user machine sends a message to the database machine. At this time, the user’s pseudo-agent is assigned to a real agent, if one is available. If none is available at the moment, that user’s pseudo-agent is placed in a first-in, first-out wait queue. When a real agent becomes available, the first pseudo-agent in the wait queue is assigned to that real agent.

When the user performs any action that results in the end of an LUW (for example COMMIT or ROLLBACK), that user’s pseudo-agent is deallocated from the real agent. An exception occurs when there are no waiting pseudo-agents and the user has sent another message to the database machine. When a pseudo-agent is deallocated from a real agent, it is placed on an inactive queue until and unless the user sends another message. At that time the pseudo-agent is placed at the end of the wait queue, unless a real agent is available. When a pseudo-agent is deallocated from a real agent, the first waiting pseudo-agent on the wait queue is allocated to the available real agent.

A pseudo-agent is deallocated from a user when the connection to the database manager is severed (for example, COMMIT WORK RELEASE or end-of-program).

The database manager does not verify that the users are allocated to real agents; that is, it does not determine whether the real agent has received a message recently (is active). A user can tie up a real agent by being inactive. For example, a user can start an LUW and leave the terminal unattended. In this situation, the DB2 Server for VM operator can use the FORCE command to end the LUW.

Pseudo-agents that are not attached to real agents have no effect on performance other than the use of extra virtual storage.

Pseudo-agents can affect shutdown procedures. When the DB2 Server for VM operator issues any SQLEND command (except SQLEND QUICK), the database manager does not end (or begin the archive process) until all users (owners of pseudo-agents) are disconnected. All users can complete their work and disconnect from the database manager (unless forced off by the VM system operator or by the DB2 Server for VM operator).

You can determine inactive but connected users by issuing the `SHOW USERS` command. The `SHOW ACTIVE` command is inappropriate because it displays information about agent structures. It does not tell you whether inactive users are holding pseudo-agents.

Note: The DB2 Server for VM operator can force (with the `FORCE` command) only those users attached to real agents. Only the VM system operator can force (log off) those users who are waiting for real agents or who have inactive pseudo-agents. The alternative is for the DB2 Server for VM operator to issue the `SQLEND QUICK` command, which immediately stops the application server and disconnects all users.

In some situations, you may want to limit the number of users who can connect to the database manager. For example, if your installation has 100 DB2 Server for VM users, you may want only 50 of them on at a time for performance reasons. Lower the `MAXCONN` parameter to decrease the number of users. This places a limit on the number of connected users. Users who try to access the database manager when the limit is reached receive a message indicating that they cannot access the database.

Application Program Use of APPC/VM or IUCV

The database manager's use of APPC/VM does not preclude users from using both APPC/VM and SQL statements in the same application program in either single user mode or multiple user mode.

For more information about how the database manager uses APPC/VM and IUCV, see the *DB2 Server for VM Diagnosis Guide and Reference* manual.

Chapter 5. Operating the Online Support for VSE Guest Sharing

This chapter explains how to enable VSE guests to access an application server on a VM operating system, and how to operate the VSE online support.

Operating VSE Guest Sharing

Your VSE online users can access an application server on a VM host operating system when the VSE operating system is running as a guest in a virtual machine. Database switching is supported for CICS online applications, which means that one resource adapter in one CICS region can connect to multiple application servers. Any CICS transaction in the CICS region can connect to any of the DB2 Server for VM application servers to which the online resource adapter has established connections. This means that:

1. Different transactions in a CICS region will be able to connect to different DB2 Server for VM application servers
2. Single transactions will be able to connect to different DB2 Server for VM application servers in different units of work.

The DB2 Server for VM application server can be accessed by specifying the `server_name` parameter on the CIRB transaction or on the CIRA transaction. The DB2 Server for VM application server can be listed in the DBNAME Directory. The DBNAME Directory provides the mapping of mapped DBNAME to *resid*; if the *resid* is different from the DBNAME, a DBNAME Directory entry must exist. See *DB2 Server for VSE System Administration* for more DBNAME Directory information. The *resid* is the basic DBNAME, and must be the same as the one specified in the SET APPCVM command during the VSE initial program load. If there are multiple DB2 Server for VM servers on the VM host, there can be more than one SET APPCVM command.

The VM application server being accessed can be either on the same processor or on another processor in the network. For batch applications and for online users who want to access an application server on another processor in a SNA network, you must issue the SET APPCVM command when you start VSE. The command provides routing information for both batch and online users. Note that SET APPCVM is required only if VTAM is to be used in the connection. If the server and requester are in a TSAF collection on the same node, it is not necessary to issue the SET APPCVM command.

Figure 19 shows the syntax of the SET APPCVM command.

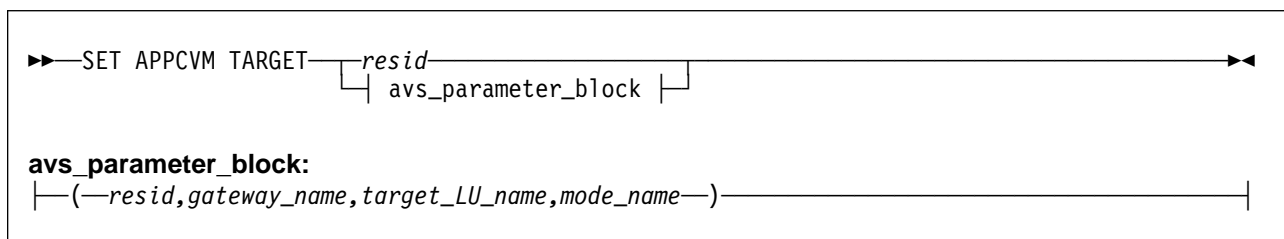


Figure 19. SET APPCVM Command

The variables have the following meanings:

resid

The resource identifier of the DB2 Server for VM application server which is the same as the *resid* parameter on the IUCV *IDENT entry in the database machine directory for VM operating systems.

avs_parameter_block

Only specify these parameters if the application server you want to access is in an SNA network. The names are defined by VTAM statements when the network is built, and have these meanings:

resid

The resource identifier of the DB2 Server for VM application server. This is the same as the *resid* parameter on the IUCV *IDENT entry in the database machine directory on VM.

gateway_name

This corresponds to an APPL statement at the local system. To the SNA network, *gateway_name* is an LU with the same name.

target_LU_name

This corresponds to an APPL statement at the remote system.

mode_name

This corresponds to a mode table entry at the local and remote systems.

The parameters **must** be specified in the order shown above.

For more information about the AVS parameters, see the *VM/ESA: Connectivity Planning, Administration, and Operation* manual. For more information on the IPL SET APPCVM command in VSE, see *VSE/ESA System Control Statements*.

Note: The VSE Guest sharing facility requires 40KB of real storage for each database communication link. For more information on providing real storage, see *VSE/ESA System Control Statements*

Operator Responsibilities

VSE guest sharing is monitored from the VM console. All DB2 Server for VM operator commands can be used. In addition, in-doubt LUWs can be forced from the VM console.

VSE online support is needed if the VSE guest is using ISQL or a CICS transaction program. The DB2 Server for VSE online resource adapter must be started so that the application server can be accessed from the CICS online environment. If this is not done, and a CICS transaction attempts to access the application server, CICS will end the transaction with CICS/VSE abend code AEY9.

Operation of the online support involves the following:

1. Starting the application server in multiple user mode, either before or after CICS is started.
2. Starting the DB2 Server for VSE online support by running the CIRB transaction under CICS. The CIRB transaction accepts a list of server names. This allows online access to multiple application servers to be established from one command. After CIRB has successfully completed its processing, the

online resource adapter is ready to handle SQL requests from CICS transaction programs (such as ISQL).

3. After the online resource adapter is started, the CICS transaction CIRA can be used to add connections or enable online access to other application servers. CIRA can be entered multiple times with different *server_names*. This establishes the connections or enables online access to the specified application server. CIRA also accepts a list of *server_names* so that online access to multiple servers can be established with one command.
4. The transaction CIRR can be used to remove connections or disable online access to a particular application server or list of application servers. The online resource adapter is terminated if the CIRR transaction removes the connection or disables online access to the last application server.
5. Displaying information about active CICS transactions (including ISQL) that access an application server by using the CIRD transaction. The CIRD transaction accepts a *server_name* parameter to display the transactions accessing a particular application server. The * keyword can be specified to display all transactions on all of the application servers (for example, CIRD *).
6. Changing the default application server using the CICS transaction CIRC.
7. Stopping the online support without stopping either CICS or the application server by issuing the CIRT transaction. The CIRT transaction terminates all connections or access to all application servers and then terminates the online resource adapter.

If a local application server becomes unavailable for some reason, only the connections to that application server are lost. The online resource adapter remains active and connections or online access to other application servers can still be used. When the local application server becomes available again, the CIRA transaction can be used to re-establish connections to it. If there are any in-doubt LUWs associated with this application server, they will be resolved at this time.

If the default application server becomes unavailable, a new default server is not established automatically. Users attempting to connect to the default server will receive a message indicating that the server is not available.

These steps are described in detail below. For more information on starting and stopping online support for VSE guest sharing, see the *DB2 Server for VSE & VM Operation* manual.

Starting the Application Server

Start the application server in multiple user mode from the DB2 Server for VM console. (Online environment is not supported in single user mode.) Next, load VSE and use the SET APPCVM command to identify the application server to VSE. For more information on the APPCVM command, see “Operating VSE Guest Sharing” on page 103. If this is the first time CICS is started, you must grant SCHEDULE authority to DBDCCICS on the application server.

Starting the Online Resource Adapter -- The CIRB Transaction

To activate the online support, run the CIRB transaction. When it completes, the resource adapter is enabled. Only when this happens can user transactions be executed.

CIRB has six parameters:

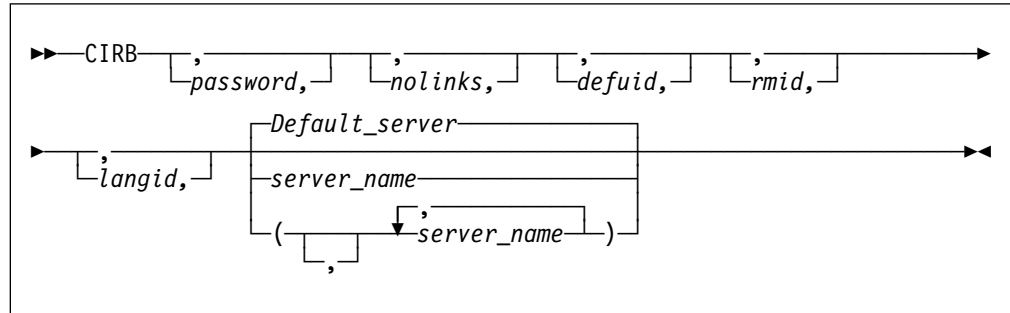


Figure 20. CIRB Transaction Syntax

The parameters are described in the following table:

Parameter	Default	Description
PASSWORD (positional parameter 1)	SQLDBAPW	This parameter establishes the operator's authority to activate online access to a local application server. The password identifies the CICS subsystem. The user ID of the subsystem is the CICS APPLID, which defaults to DBDCCICS. The procedure ARIS080D uses the following job control to give the password and user ID to the local application server: <pre>// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S, LOGMODE=N,PROGNAME=ARIDBS' CONNECT SQLDBA IDENTIFIED BY SQLDBAPW; GRANT SCHEDULE TO DBDCCICS IDENTIFIED BY CICSPSWD; COMMIT WORK;</pre> The password chosen (CICSPSWD above) must satisfy DB2 Server for VSE & VM specifications for a password. This password establishes which password to use when dropping connections through the CIRR or CIRT commands. See "Password Implications on Online Resource Adapter Termination" on page 140 for more details.
NOLINKS (positional parameter 2)	3	This parameter establishes the number of links (paths) that should be initialized to a local application server. Specify this parameter as a decimal value between 1 and 64. The number must be less than or equal to the value assigned to the NCUSERS initialization parameter of the DB2 Server for VSE & VM system. (The NCUSERS default is 5.)
DEFUID (positional parameter 3)	CICSUSER	This parameter identifies the default user ID used by the online support when it makes an implicit CONNECT to a local application server. This parameter must satisfy DB2 Server for VSE & VM specifications for a user ID.

<i>Table 8 (Page 2 of 3). CIRB Transaction Parameters</i>		
Parameter	Default	Description
RMID (positional parameter 4)	0	<p>This parameter identifies a unique resource adapter. You must specify it only if your installation has multiple CICS partitions active in the same VSE/ESA system, and if each CICS partition allows online access to a server. For the case of a local application server, recovery requires that the local server know the resource adapter it is servicing. You must specify this parameter as a decimal value between 0 and 63.</p> <p>If the DB2 Server for VSE online support detects that this ID is not unique in the system, it issues a message. The CIRB transaction then ends without enabling the resource adapter.</p> <p>There can be only one DB2 Server for VSE resource adapter enabled in a single CICS partition. An attempt to enable a second DB2 Server for VSE resource adapter causes the DB2 Server for VSE online support to issue a message, and the CIRB transaction ends without enabling the second resource adapter. The first one, however, remains in effect.</p>
LANGID (positional parameter 5)	specified at installation	<p>This parameter defines the language the DB2 Server for VM online support uses to display error and information messages. The language you specify on this transaction becomes the default language for ISQL, CBND, C2BD, DSQG, DSQU, DSQD, and DSQQ. The ISQL welcome logo always appears in the language specified on this transaction.</p> <p>This parameter must take the form of a minimum 1-character, maximum 5-character language ID. You must use one of the language IDs in the LANGID column of the SQLDBA.SYSLANGUAGE table. The language ID must identify a language you have installed on the DB2 Server for VM server. To choose another language, use the SET LANGUAGE command in ISQL. The following IDs can be specified on the CIRB transaction:</p> <p>AMENG American English UCENG Uppercase English FRANC French GER German KANJI Kanji (Japanese) HANZI Simplified Chinese</p> <p>If this parameter is omitted, the language defaults to the language chosen as the default at installation.</p>

<i>Table 8 (Page 3 of 3). CIRB Transaction Parameters</i>		
Parameter	Default	Description
SERVER-NAME (positional parameter 6)	Determined from DBNAME directory or "SQLDS."	<p>This parameter enables you to specify the application servers that you want to access. If the list format specifies multiple servers, the first one in the list becomes the default server. Only the first server_name in the list may be omitted.</p> <p>If this parameter (or the first one in the list) is omitted, the default server is determined from the DBNAME directory. If the DBNAME directory does not specify a default server, then SQLDS becomes the default server name.</p> <p>If the application server specified is in the DBNAME Directory and the corresponding SYSID is present in the DBNAME Directory, this server is considered a remote application server. Otherwise, it is considered as a local application server.</p> <p>If the application server specified is not in the DBNAME Directory and the length of the server name does not exceed 8 characters, the online support will access this application server via guest sharing.</p>

The CIRB transaction establishes the default application server. If the server_name parameter is not specified on the CIRB transaction, then the default server is determined from the DBNAME directory. If a single server_name is specified on the CIRB transaction then it becomes the default server. If a server_name list is specified on the CIRB transaction, the first server_name in the list becomes the default server. If the first server_name in the server_name list is blank then the default server is determined in the same way as when the server_name is omitted from the CIRB transaction. For example:

```
CIRB ,,,, (,SQLMACH2)
```

This starts connections to two servers. The first one is the default server and its name is determined from the DBNAME directory or if it is not specified in the DBNAME directory it defaults to **SQLDS**. The second server is SQLMACH2.

Note that the following examples are not allowed. Only the first server_name in the list can be blank.

```
CIRB ,,,, (SQLMACH2,)  
CIRB ,,,, (SQLMACH2,,SQLVM)
```

The number of server_names that can be specified on the CIRB command is limited by the size of the input line on the VSE console or a CICS terminal. The VSE console only allows one line of input. A CICS terminal allows much more input. If short server_names are used more can fit on the command. Server-names can be up to 18 characters long. If all of the required server_names cannot fit on the command, the CIRA transaction must be used to establish connections for the remaining server_names.

Figure 21 on page 109 shows an example of using the server_name list on the CIRB transaction.

```

msg f2
AR 015 1I40I READY
2 cirb ,,,, (sqlmach1,sqlmach1)
F2-002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2-002 ARI0450I DB2 Server for VSE online support has an
           entry point of 003AA808 RMGL at 00541200.
F2-002 ARI0454I Connections to SQLMACH1 established.
           RMCV at 0055B2E0.
F2-002 ARI0458I The default server is SQLMACH1.
F2-002 ARI0457W Connections to SQLMACH1 already exist.
F2-002 ARI0402E Connections to SQLMACH1 could not be established.

```

Figure 21. Example of CIRB with Duplicate Server Names

The maximum number of application servers to which an online resource adapter can establish connections or enable online access to is only limited by the amount of storage available in the partition where the online resource adapter is running.

If you try to establish connections to an application server to which connections already exist, or to which online access is already enabled, the message “ARI0457W Connections to <server_name> already exist.” is displayed. No action is taken against that server. If the connections to a local server need to be changed they must first be removed using CIRR or CIRT and then re-established using CIRA or CIRB. An example is shown in Figure 22.

```

msg f2
AR 015 1I40I READY
2 cirb ,,,, (sqlmach1,sqlmach2)
F2-002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2-002 ARI0450I DB2 Server for VSE online support has an
           entry point of 003AA808 RMGL at 00541200.
F2-002 ARI0454I Connections to SQLMACH1 established.
           RMCV at 0055B2E0.
F2-002 ARI0458I The default server is SQLMACH1.
F2-002 ARI0454I Connections to SQLMACH2 established.
           RMCV at 0055C2E0.
2 cirr ,,, sqlmach2
F2-002 ARI0455I Connections to SQLMACH2 are disabled.
2 cira ,5,, sqlmach2
F2-002 ARI0454I Connections to SQLMACH2 established.
           RMCV at 0055A2E0.

```

Figure 22. Example of Changing Connection Settings

Note that each local server in the list has its connections established with the same values for password, number of links, RMID, default user ID and language ID that were specified.

If the CIRB parameters for each server are identical, all of the connections or online access can be established with one CIRB transaction, as illustrated in Figure 23 on page 110.

```

msg f2
AR 015 1140I READY
2 cirb ,,,, (sqlmach1,sqlmach2,sqlvm)
F2-002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2-002 ARI0450I DB2 Server for VSE online support has an
           entry point of 003AA808 RMGL at 00541200.
F2-002 ARI0454I Connections to SQLMACH1 established.
           RMCV at 0055A2E0.
F2-002 ARI0458I The default server is SQLMACH1.
F2-002 ARI0454I Connections to SQLMACH2 established.
           RMCV at 0055C2E0.
F2-002 ARI0454I Connections to SQLVM established.
           RMCV at 0055D2E0.

```

Figure 23. Example of CIRB with Server-Name List

All three local application servers have the same number of connections, the same default user ID, the same password, the same RMID and the same language ID.

If one or more of the parameters must be different, then all of the connections cannot be established with one CIRB transaction. You will need the CIRA transaction to add additional servers.

If you enter a remote server name in the server_name parameter of the CIRB or CIRA transaction, CIRB or CIRA will not establish any links or sessions to the remote system where the remote server runs. The following message will not be displayed by CIRB or CIRA when it is processing a remote server, but will display for local servers.

```

ARI0454I Connections to server_name established.
           RMCV at XXXXXXXX.

```

CIRB or CIRA will display the following message instead for every remote server processed at initialization time:

```

ARI0467I RMCV for remote server_name established.
           RMCV at XXXXXXXX.

```

Starting the CIRB Transaction

The CICS sequential device support can be used to automatically start the CIRB transaction when CICS is started. Either a CRLP (a card reader or line printer) device, or a sequential DASD device must be defined in the CICS DFHTCT, to allow them to simulate terminals.

If a CRLP device is defined, the CIRB transaction can be run automatically by including it in the CICS startup jobstream. The CIRB statement should be coded just as it would if it were entered from a terminal. Include a slash (\) at the end of the statement to indicate the end of data. Figure 24 shows an example:

```

// EXEC DFHSIP,SIZE=NNNNK
CIRB PASSWORD,3,PRODCICS,0\
/*

```

Figure 24. Automatically Starting CIRB

If a sequential DASD device has been defined in the CICS DFHTCT, you must define two sequential DASD data sets: one input and one output. These can be

either sequential access method (SAM) data sets or SAM-managed VSAM data sets. The input data set must contain the CIRB statement. (A utility such as DITTO or VSAM IDCAMS can be used to load the CIRB statement to the data set.) The output data set will contain the messages from the CIRB startup process. Whichever type of device is used -- CRLP or DASD -- do not include a CSSF GOODNIGHT statement following the CIRB statement, as this would allow the statement to be processed in all CICS startup modes (cold, auto, and emer).

The application server must be started before CICS for automatic startup to work. When the CIRB transaction successfully ends, the following message is displayed at the VSE console:

```
ARI0410I Resource Adapter ARI00LRM is enabled
```

For more information about CICS sequential device support, see the *CICS/VSE Resource Definition (Macro)* manual. For information about the DFHTCT entries required to define a sequential CRLP or DASD device, see the *DB2 Server for VM Program Directory*.

If a failure occurs, you can issue the CIRT transaction with the QUICK mode. This mode disconnects links to the application server. For more information, see “Stopping the Online Support -- The CIRT Transaction” on page 135. If the above action does not solve the problem, CICS must be recycled.

SCHEDULE Authority for VSE Guest Sharing

The VM database must grant SCHEDULE authority to the CICS/VSE application identifier.

Implicit CONNECT Support

This support allows development of online applications that do not issue an SQL CONNECT statement. With this support, operators need not enter a user ID and password as input to the online application, which is useful if your installation requires terminal users to sign on using the CSSN transaction. For some transactions accessing the database, the CICS sign-on verification may be sufficient. It can also be useful if you have just installed the database manager and find it convenient to have all users identified by one name (for example, CICSUSER).

If a CICS transaction has not yet established a user ID for the current or prior unit of work, and the user has signed on to CICS using the CESN (or CSSN) transaction, online support will attempt to use the eight-character signon user ID. The user ID used will be the value returned by the CICS command

```
EXEC CICS ASSIGN USERID(data-area)
```

If you start the online support with CIRB, then before the online resource adapter is able to run the implicit connect support to a local application server, it verifies that the CICS subsystem has SCHEDULE authority on the local application server.

Grant the necessary SCHEDULE authority as follows:

```
GRANT SCHEDULE TO CICSTEST IDENTIFIED BY cicspw
```

where *cicspw* is the new password. The required password input parameter for CIRB (and CIRT) is now *cicspw*.

If the online support can verify that the CICS subsystem has SCHEDULE authority, it sets the DEFUID into each of the agents allocated for online use. The DEFUID you specify as an input parameter for CIRB is the user ID used for all online applications connecting to a local application server that do not issue an SQL CONNECT statement and do not have a valid CICS signon user ID.

Supporting Multiple User Online Access

The NOLINKS input parameter to CIRB causes the allocation of a fixed number of links to the local application server. The online support suballocates the links to CICS transactions when they issue their first SQL request. When a transaction has a link, it keeps it until the end of the logical unit of work. When the number of such transactions exceeds NOLINKS, some transactions have to wait for links, and link contention occurs. Some planning is required to optimize the NOLINKS parameter. NOLINKS varies as your application mix varies.

Consider these things about the NOLINKS input parameter:

- Initially, allow one link for each one to two ISQL users, and one link for each four to ten users of preplanned transactions.
- The NOLINKS value must not exceed that of the NCUSERS initialization parameter, which defines the total number of links to the application server.
- The online support uses the CICS monitoring facility to collect performance data. For a given NOLINKS and a given period of the day, you can gather information on the number of link waits, total link wait time, and total time holding the link. For more information, see the *DB2 Server for VSE & VM Performance Tuning Handbook*.
- When a logical unit of work ends, the online support makes the freed link available to all waiting transactions. The first waiting database transaction that CICS dispatches gets the link. To define allocation priority for the online links, consider using the operator, transaction, and terminal priority mechanisms of CICS. (These are specified with the OPPRTY keyword of DFHSNT, and the TRMPRTY keyword of DFHTCT respectively.)
- Consider defining one or more transaction classes for the transactions that access the database manager, and limit access by using the CICS CMXT keyword of DFHSIT. By correlating CMXT with NOLINKS, you can ensure that storage resources in the CICS partition are not used until links are available.
- Consider a similar technique to control the number of active ISQL users. Rather than limit the total number of active ISQL users, you can control the number of active users from a given department or user group. See “Access Control to ISQL on a VSE Guest” on page 149.

CIRB Impact to System Resources

If the NOLINKS input parameter is n , system resources are used as follows:

- You have n links allocated to the application server, and n application server agents are used. The agents remain allocated for online applications until CIRT is entered.
- Additional virtual storage is required in the CICS partition for the online support. See Appendix A, “Virtual and Real Storage Requirements” on page 443.

- For each concurrent transaction that is attempting to access the application server, additional virtual storage is required in the CICS partition. See Appendix A, “Virtual and Real Storage Requirements” on page 443.

Supporting Multiple CICS Partitions

Your installation can have multiple CICS partitions, each with access to the application server. For recovery purposes, each instance of an active online resource adapter must have a unique identifier. You can do this with the CIRB RMID input parameter. You should keep the RMID for a CICS partition consistent, by relating the RMID to the priority of each CICS, specifying a 0 for the production CICS, 1 for the test-level CICS, and so on. If your installation has only one CICS system, the RMID input parameter need not be specified.

Adding Connections -- The CIRA Transaction

The CIRA transaction has four parameters:

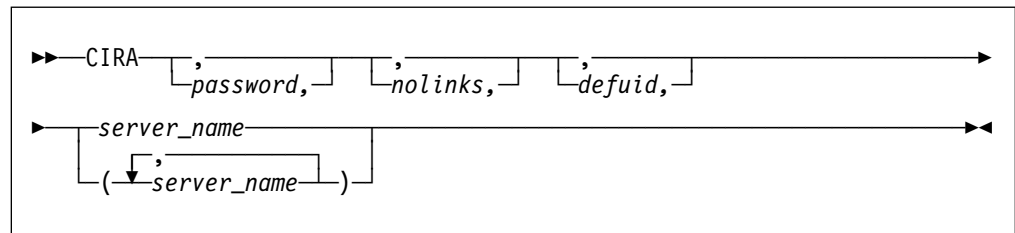


Figure 25. CIRA Transaction Syntax

The parameters are described in the following table:

Parameter	Default	Description
PASSWORD (positional parameter 1)	SQLDBAPW	<p>This parameter establishes the operator's authority to activate online access to a local application server. The password identifies the CICS subsystem. The user ID of the subsystem is the CICS APPLID, which defaults to DBDCCICS. The procedure ARIS080D uses the following job control to give the password and user ID to the DB2 Server for VM server:</p> <pre>// EXEC ARISQLDS,SIZE=AUTO,PARM='SYSMODE=S, LOGMODE=N,PROGNAME=ARIDBS' CONNECT SQLDBA IDENTIFIED BY SQLDBAPW; GRANT SCHEDULE TO DBDCCICS IDENTIFIED BY CICSPSWD; COMMIT WORK;</pre> <p>The password chosen (CICSPSWD above) must satisfy DB2 Server for VSE & VM specifications for a password. This password establishes which password to use when dropping connections through the CIRR or CIRT commands. See "Password Implications on Online Resource Adapter Termination" on page 140 for more details.</p>
NOLINKS (positional parameter 2)	3	<p>This parameter establishes the number of links (paths) that should be initialized to a local application server. Specify this parameter as a decimal value between 1 and 64. The number must be less than or equal to the value assigned to the NCUSERS initialization parameter of the DB2 Server for VSE & VM system. (The NCUSERS default is 5).</p>
DEFUID (positional parameter 3)	CICSUSER	<p>This parameter identifies the default user ID used by the online support when it makes an implicit CONNECT to a local application server. This parameter must satisfy DB2 Server for VSE & VM specifications for a user ID.</p>
SERVER-NAME (positional parameter 4)	none	<p>This parameter is required and it specifies the additional application servers (local or remote), that you want to access.</p> <p>If this parameter is omitted, the message ARI0400E is issued indicating that an invalid input parameter was entered.</p>

The password, nolinks, defuid and server_name parameters have exactly the same meanings as on the CIRB command. One exception is that the server_name parameter is required on CIRA but is optional on CIRB.

The number of server_names that can be specified on the CIRA command is limited by the size of the input line. As with CIRB, CIRA can be entered on the VSE console or on a CICS terminal. On the VSE console the input is limited to one line. On the CICS terminal it can use the full screen. If short server_names are used more can fit on the command. Server_names can be up to 18 characters long. If all of the required server_names cannot fit on the command, the CIRA transaction must be repeated for the remaining server_names. Figure 26 on page 115 shows an example using the CIRA transaction with a server_name list.

```

msg f2
AR 015 1I40I READY
2 cirb ,,,,sqlmach1
F2-002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2-002 ARI0450I DB2 Server for VSE online support has an
           entry point of 003AA808 RMGL at 00541200.
F2-002 ARI0454I Connections to SQLMACH1 established.
           RMCV at 0055A2E0.
F2-002 ARI0458I The default server is SQLMACH1.
2 cira ,,,(sqlmach2,sqlvm)
F2-002 ARI0454I Connections to SQLMACH2 established.
           RMCV at 0055C2E0.
F2-002 ARI0454I Connections to SQLVM established.
           RMCV at 0055D2E0.

```

Figure 26. Example of CIRA with Server_Name List

The maximum number of application servers to which an online resource adapter can establish connections or enable online access to is only limited by the amount of storage available in the partition where the online resource adapter is running.

The CIRA transaction establishes connections or enables online access to the specified application servers based on the parameters given on the CIRA transaction. If a server_name list is used then connections or online access will be established to each application server in the list using the same set of parameters. For example:

```
CIRA thispw,4,thisid,(sqlmach2,sqlvm)
```

The above command will establish four links to the local application server SQLMACH2 with password "thispw" and default user ID "thisid." The RMID and the language ID are inherited from the CIRB transaction. If the online resource adapter was started with RMID = 0 and language ID = ameng then any connections started to that same online resource adapter will also have RMID = 0 and language id = ameng. Then CIRA will establish four links to SQLVM with password "thispw" and default user ID "thisid." Again the RMID is 0 and the language ID is ameng. If CIRA is entered before CIRB was run, the message "ARI0411I Resource Adapter is not enabled." is displayed.

If one or more of the parameters must be different, then the server_name list format of the CIRA transaction cannot be used. The CIRA transaction would have to be executed separately for each application server that required different parameters. For example, if three links are required to SQLMACH2 and four links are required to SQLVM but the other parameters are the same for both servers, the CIRA transaction must be run for each of them.

```
CIRA thispw,3,thisid,sqlmach2
CIRA thispw,4,thisid,sqlvm
```

If you try to establish connections or enable online access to an application server that is already connected a warning message will be displayed. No action is taken against that server. If the connections to a local application server need to be changed they must first be removed using CIRR or CIRT and then re-established using CIRA or CIRB.

Consider the following scenario. An online transaction program needs to access three different application servers, SQLMACH2, SQLMACH1 and SQLVM.

SQLMACH2 and SQLMACH1 are running in two VSE partitions and SQLVM is running under VM and is accessed via guest sharing. We want SQLMACH1 to be the default server, and we want the default settings for all three servers.

To achieve this we could enter the following sequence of commands. Assume that our CICS region is running in partition 2, SQLMACH2 is running in partition 4 and SQLMACH1 is running in partition 5.

1. Use the CIRB transaction to start the online resource adapter and establish the default application server, SQLMACH1.
2. Use the CIRA transaction to establish connections to SQLMACH2.
3. Use the CIRA transaction again to establish connections to SQLVM.

This is illustrated in Figure 27.

```
F2-002 DFH1500 - DBDCCICS : CONTROL IS BEING GIVEN TO CICS
msg f2
AR 015 1I40I READY
2 cirb ,,,,sqlmach1
F2-002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2-002 ARI0450I DB2 Server for VSE online support has an
           entry point of 003AA808 RMGL at 00541200.
F2-002 ARI0454I Connections to SQLMACH1 established.
           RMCV at 0055D2E0.
F2-002 ARI0458I The default server is SQLMACH1.
2 cira ,,,sqlmach2
F2-002 ARI0454I Connections to SQLMACH2 established.
           RMCV at 0055C2E0.
2 cira ,,,sqlvm
F2-002 ARI0454I Connections to SQLVM established.
           RMCV at 0055A2E0.
```

Figure 27. Example of CIRB and CIRA

Since the settings for the connections to SQLMACH2 and SQLVM are identical, both connections could be established on the same CIRA command, as illustrated in Figure 26 on page 115.

Automatic Restart Resynchronization

If a system or subsystem failure occurs while an online application is trying to commit work and two-phase commit is being used, the unit being committed is called an in-doubt logical unit of work, because the database manager has prepared it for commit or rollback but the system or subsystem failure occurred before the commit completed. In-doubt units of work must be resolved the next time the application server is started.

Note: CICS/VSE and the local application server will use a one-phase commit if at most one external resource has been updated. In this case it is not possible to create an in-doubt unit of work. This means that any CICS transaction that updates only the local application server resources will not generate in-doubt units of work.

The CICS/VSE restart resynchronization facility, which is started implicitly when you issue CIRB or CIRA, resolves the in-doubt units of work created by any CICS

transaction that updated a local application server. To enable it, you must update the CICS/VSE tables to include the resynchronization transaction.

CIRB and CIRA assume that restart resynchronization is enabled when they are executed. If, for some reason it has been disabled when CIRB or CIRA is issued, it will display the message *"ARI0466E CICS restart re-synchronization is not available. The <tran> transaction is ended."* and exit. At this point the system programmer should ensure that it has been properly enabled and retry CIRB or CIRA.

For information about the updates, see the *DB2 Server for VSE Installation* manual.

The current implementation of the CICS/VSE restart resynchronization facility allows it to re-synchronize itself with DB2 Server for VSE online resource adapter only once. After it has been invoked, CICS discards any information about in-doubt units of work that it did not resolve. This means that there can be scenarios where it is not possible to automatically resolve in-doubt units of work.

When the CIRB or CIRA transaction is started, a connection is made to the READY/RECOVERY agent of the local server to get a 'recovery list'. This recovery list provides information on any in-doubt agents that need to be resolved for this server. After this has been done for every local server specified in the CIRB or CIRA command, the CICS/VSE restart resynchronization facility is invoked, which will resolve the in-doubt units of work for all of those local servers. A subsequent CIRA to connect to another local server that also has in-doubt units of work will fail because CICS has discarded the log information. The in-doubt units of work on that server must be resolved manually using the FORCE n COMMIT or FORCE n ROLLBACK commands on the server before the CIRA command will work.

For example, suppose that SQLMACH1 and SQLMACH2 are DB2 Server for VM application servers that run on the same VM system and are accessed via guest sharing. The password used to access SQLMACH1 is ABC and the password used to access SQLMACH2 is DEF. All the other parameters needed by the two databases are the defaults. The connections to SQLMACH1 and SQLMACH2 are established using the following sequence of commands:

```
CIRB abc,,,,sqlmach1
```

```
CIRA def,,,,sqlmach2
```

Suppose that CICS transactions accessing these application servers also make updates to the DB2 Server for VM database as well as some other external non-CICS resource, so that CICS will use the two-phase commit process. If a system failure occurs on the VM system while CICS is performing a two-phase commit to both these databases, then both SQLMACH1 and SQLMACH2 will go down. When the system is brought back up and SQLMACH1 and SQLMACH2 are restarted, they will both have in-doubt units of work. If the connections to SQLMACH1 and SQLMACH2 are restarted the same way as before, only the in-doubt units of work on SQLMACH1 will be resolved automatically. The in-doubt units of work on SQLMACH2 will need to be resolved explicitly before the CIRA command for SQLMACH2 will work.

See Figure 28 on page 118 for an example of this.

```

2 cirb abc,,,,,sqlmach1
F2 002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2 002 ARI0450I DB2 Server for VSE online support has an
           entry point of 0039F008 RMGL at 001DF5B4.
F2 002 ARI0454I Connections to SQLMACH1 established.
           RMCV at 0053BF00.
F2-002 ARI0458I The default server is SQLMACH1.
2 cira def,,,sqlmach2
F2-002 ARI0454I Connections to SQLMACH2 established.
           RMCV at 0055A080.

<System Failure occurs>

F2 002 ARI2908I XPCCB, IJBXRUSR = 0483061009000000
F2 002 ARI0406E Error in using system communications facility.
           Request = 15
           Return Code = 4 Reason Code = 7
F2 002 The default server is SQLMACH1.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH1.
F2 002 Status of DB2 Server for VSE online applications:
F2 002
F2 002 Transactions holding a link to the application server but not using are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID    USERDATA  TIME SINCE  TOTAL LUW
F2 002                                     LAST ACCESS TIME
F2 002 -----
F2 002 0000041  CISQ    SQLDBA    L080      00:00:06  00:01:34
F2 002
F2 002 TIME= 15:26:15 DATE= 08/14/95
F2 002 ARI0465I Transactions are still active
           for server SQLMACH1.
F2 002 ARI0463I The DISABLE transaction CIRR must delay for a
           30-second interval before attempting the disable.
F2 002 ARI0455I Connections to SQLMACH1 are disabled.
F2 002 ARI0460W Connections to the default server SQLMACH1
           have been disabled.
F2 002 ARI2908I XPCCB, IJBXRUSR = 0483061009000000
F2 002 ARI0406E Error in using system communications facility.
           Request = 15
           Return Code = 4 Reason Code = 7
F2 002 The default server is SQLMACH1.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH2.
F2 002 Status of DB2 Server for VSE online applications:
F2 002
F2 002 Transactions holding a link to the application server but not using are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID    USERDATA  TIME SINCE  TOTAL LUW
F2 002                                     LAST ACCESS TIME
F2 002 -----
F2 002 0000141  CISQ    SQLDBA    L083      00:00:06  00:01:34
F2 002

```

Figure 28 (Part 1 of 2). Automatic Restart Resynchronization Failure


```

F2 002 TIME= 15:26:45 DATE= 08/14/95
F2 002 ARI0465I Transactions are still active
           for server SQLMACH2.
F2 002 ARI0463I The DISABLE transaction CIRR must delay for a
           30-second interval before attempting the disable.
F2 002 ARI0455I Connections to SQLMACH2 are disabled.
F2-002 ARI0413I Resource Adapter ARI00LRM is disabled.

<SQLMACH1 and SQLMACH2 are restarted>

2 cirb abc,,,,,sqlmach1
F2 002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2 002 ARI0450I DB2 Server for VSE online support has an
           entry point of 0039F008 RMGL at 001DF5B4.
F2 002 ARI0454I Connections to SQLMACH1 established.
           RMCV at 0053BF00.
F2-002 ARI0458I The default server is SQLMACH1.
2 cira def,,,sqlmach2
F2 002 ARI0454I Connections to SQLMACH2 established.
           RMCV at 0055A080.

F2-002
F2 002 ARI0438E Automatic restart resynchronization failed.
           A logical unit of work that DB2 for VSE indicated
           needed to be resolved was not identified by
           the CICS/VSE log as needing resolution.
F2 002 ARI0423A Use the SHOW and FORCE commands to
           COMMIT or ROLLBACK the following units of work:
F2 002 ARI0424I User ID = SQLDBA Agent Identifier = 1
           Server = SQLMACH2
F2 002 The default server is SQLMACH1.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH2.
F2 002 There are no active DB2 Server for VSE transactions.
F2 002
F2 002 TIME= 15:33:22 DATE= 08/14/95
F2 002 ARI0455I Connections to SQLMACH2 are disabled.

<From the SQLMACH2 console enter:>
<SHOW ACTIVE>
<FORCE 1 ROLLBACK>

<Now CIRA will work>

2 cira def,,,sqlmach2
F2 002 ARI0454I Connections to SQLMACH2 established.
           RMCV at 0055A080.

```

Figure 28 (Part 2 of 2). Automatic Restart Resynchronization Failure

However if the connections to SQLMACH1 and SQLMACH2 are established with a single CIRB or CIRA command, the in-doubt units of work on **both** servers will be resolved automatically.

See Figure 29 on page 120 for a detailed example of this.

```

2 cirb abc,,,,(sqlmach1,sqlmach2)
F2 002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2 002 ARI0450I DB2 Server for VSE online support has an
          entry point of 0039F008 RMGL at 001DF5B4.
F2 002 ARI0454I Connections to SQLMACH1 established.
          RMCV at 0053BF00.
F2-002 ARI0458I The default server is SQLMACH1.
F2-002 ARI0454I Connections to SQLMACH2 established.
          RMCV at 0055A080.

<System Failure occurs>

F2 002 ARI2908I XPCCB, IJBXRUSR = 0483061009000000
F2 002 ARI0406E Error in using system communications facility.
          Request = 15
          Return Code = 4 Reason Code = 7
F2 002 The default server is SQLMACH1.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH1.
F2 002 Status of online DB2 Server for VSE applications:
F2 002
F2 002 Transactions holding a link to the application server but not using are:
F2 002
F2 002  TASKNO  TRANID  TERMID  USER ID   USERDATA  TIME SINCE  TOTAL LUW
F2 002                                     LAST ACCESS TIME
F2 002  _____  _____  _____  _____  _____  _____  _____
F2 002  0000041  CISQ      SQLDBA   L080      00:00:06  00:01:34
F2 002
F2 002 TIME= 15:26:15 DATE= 08/14/95
F2 002 ARI0465I Transactions are still active
          for server SQLMACH1.
F2 002 ARI0463I The DISABLE transaction CIRR must delay for a
          30-second interval before attempting the disable.
F2 002 ARI0455I Connections to SQLMACH1 are disabled.
F2 002 ARI0460W Connections to the default server SQLMACH1
          have been disabled.
F2 002 ARI2908I XPCCB, IJBXRUSR = 0483061009000000
F2 002 ARI0406E Error in using system communications facility.
          Request = 15
          Return Code = 4 Reason Code = 7
F2 002 The default server is SQLMACH1.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH2.
F2 002 Status of online DB2 Server for VSE applications:
F2 002

```

Figure 29 (Part 1 of 2). Successful Automatic Restart Resynchronization

```

F2 002 Transactions holding a link to the application server but not using are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID    USERDATA  TIME SINCE  TOTAL LUW
F2 002                                     LAST ACCESS TIME
F2 002 _____  _____  _____  _____  _____  _____  _____
F2 002 0000141  CISQ      SQLDBA   L083      00:00:06  00:01:34
F2 002
F2 002 TIME= 15:26:45 DATE= 08/14/95
F2 002 ARI0465I Transactions are still active
                for server SQLMACH2.
F2 002 ARI0463I The DISABLE transaction CIRR must delay for a
                30-second interval before attempting the disable.
F2 002 ARI0455I Connections to SQLMACH2 are disabled.
F2-002 ARI0413I Resource Adapter ARI00LRM is disabled.

<SQLMACH1 and SQLMACH2 are restarted>

2 cirb abc,,,,,(sqlmach1,sqlmach2)
F2 002 ARI0410I Resource Adapter ARI00LRM is enabled.
F2 002 ARI0450I DB2 Server for VSE online support has an
                entry point of 0039F008 RMGL at 001DF5B4.
F2 002 ARI0454I Connections to SQLMACH1 established.
                RMCV at 0053BF00.
F2-002 ARI0458I The default server is SQLMACH1.
F2 002 ARI0454I Connections to SQLMACH2 established.
                RMCV at 0055A080.

```

Figure 29 (Part 2 of 2). Successful Automatic Restart Resynchronization

Assuming CICS restart resynchronization has been properly enabled as described in the *DB2 Server for VSE Installation* manual, the conditions where in-doubt units of work must be resolved explicitly are:

1. CICS log missing. This can be from a CICS log media failure, CICS COLD start which destroys the log contents, or CICS journal is not active so no log data is created.
2. CICS RESYNCH has already been issued. The log data is discarded by CICS after the RESYNCH command has been issued even if it was not used. See Figure 28 on page 118 for an example of this.

To take full advantage of the automatic restart resynchronization the following should be true:

1. All local application servers with in-doubt units of work must be started on the same CIRB or CIRA transaction. This means they must have the same password, default user ID, language, RMID, and number of links to be started.
2. CICS startup should be START=AUTO which lets CICS determine if the startup will be START=WARM or START=EMER. Any COLD start will erase the log data and automatic restart resynchronization will not be possible.

Resolving In-Doubt Transactions

Only under exceptional conditions (such as a CICS log media failure) do you have to resolve in-doubt LUWs explicitly. To do so, issue the SHOW ACTIVE command to determine those agents that are in-doubt; then issue the FORCE command to commit or rollback each one:

FORCE *n* COMMIT

or

FORCE *n* ROLLBACK

where *n* is the agent identifier of the in-doubt LUW.

The discussion in the *DB2 Server for VSE & VM Operation* manual states that, in general, FORCE *n* COMMIT should be entered. The exception is for applications that access multiple resources (for example, an application that updates both a DB2 Server for VM database and a VSAM/VSE file.) For such applications, the operator requires direction from the developer or user of the application.

You could plan for this situation by keeping a list of all transactions that update multiple resources. The list should contain the CICS transaction identifier for the application, and the recommended direction (COMMIT or ROLLBACK) from the developer. (For more information, see the discussion on online application recovery in the *DB2 Server for VM Database Administration* manual.) Because ISQL does not update multiple resources, the direction for the ISQL transaction should always be to commit work.

Changing the Default Server -- The CIRC Transaction

The transaction CIRC can be used to dynamically change the default server. The CIRC transaction has one parameter:

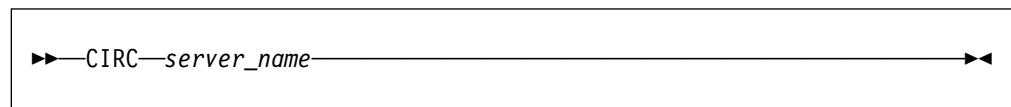


Figure 30. CIRC Transaction Syntax

The parameter is described in the following table.

Parameter	Default	Description
SERVER-NAME (positional parameter 1)	none	This parameter is required and it specifies the application server that you want to become the default. If this parameter is omitted, the message ARI0400E is issued indicating that an invalid input parameter was entered.

The server-name specified must already have connections or online access established to it, either from the CIRB or CIRA transactions. If connections to the specified server do not exist or online access to the specified server was not enabled from the CIRB or CIRA transactions, the message "ARI0456I Connections to <server-name> do not exist." is displayed. In this case the CIRA transaction must first be run to establish the connections, then the CIRC transaction is run to make it the default server.

For the following example assume that connections exist to SQLMACH1 and SQLMACH2 and that SQLMACH2 is the current default server.

```

msg f2
AR 015 1I40I READY
2 circ sqlmach1
F2-002 ARI0459I The new default server is SQLMACH1.
                The previous default server was SQLMACH2.

```

Figure 31. Example of CIRC

For this next example assume that connections exist to SQLMACH1 but not to SQLMACH2.

```

msg f2
AR 015 1I40I READY
2 circ sqlmach2
F2-002 ARI0456I Connections to SQLMACH2 do not exist.
2 cira ,,,sqlmach2
F2-002 ARI0454I Connections to SQLMACH2 established.
                RMCV at 0055D2E0.
2 circ sqlmach2
F2-002 ARI0459I The new default server is SQLMACH2. The previous
                default server was SQLMACH1.

```

Figure 32. Example of CIRC

It is important to note that if the connections to the default server are lost, or online access to the default application server is disabled, that server is still identified as the default server. The connections can be lost because the server went down or because the CIRR transaction was used to terminate the online access or connection. Users that are trying to connect to the default server in these cases will receive SQLCODE = -940. If the CIRB or CIRA transaction is used to establish connections to a local server that is not ready, the message "ARI0418A Application server <server-name> is not ready. Retry the enable transaction after the application server starts." is displayed. If the CIRB or CIRA transaction is used to establish online access to a remote server that is not ready, an error message will not be displayed. This is because CIRB or CIRA can not check whether a remote server is ready or not.

Removing Connections -- The CIRR Transaction

To remove connections or to disable online access to a local or remote application server, issue the CICS CIRR transaction. The CIRR transaction has four parameters:

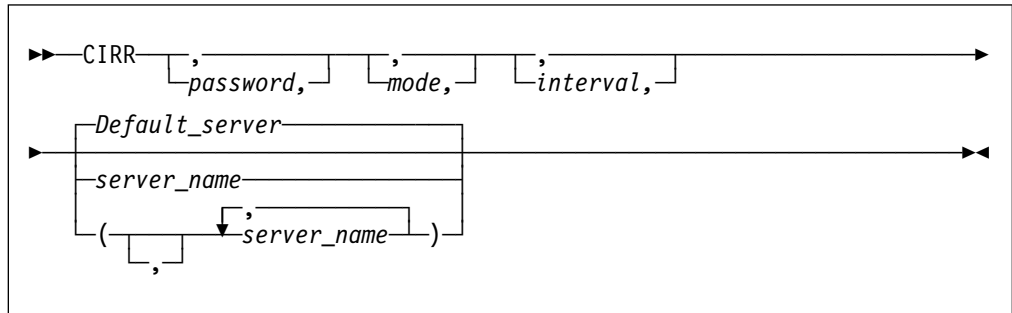


Figure 33. CIRR Transaction Syntax

The password, mode and interval parameters are the same as on the CIRT transaction and are described in the following table:

<i>Table 11. CIRR Transaction Parameters</i>		
Parameter	Default	Description
PASSWORD (positional parameter 1)	SQLDBAPW	This password establishes the operator's authority to terminate the online access to the application server. It must be the same password that was supplied for the server by the CIRB or CIRA transaction. Refer to "Password Implications on Online Resource Adapter Termination" on page 140 for more details.
MODE (positional parameter 2)	NORMAL	This parameter establishes the shutdown mode: NORMAL or QUICK. When you specify NORMAL, the CIRR transaction prevents new online users from accessing the specified application server. Users who are already doing work, however, can finish. When all users complete their work, no online users can use the specified application server. When you specify NORMAL for a remote application server, the shutdown of the access to the remote application server will complete only when all conversations to the remote application server have been deallocated. When you specify QUICK for a local application server, online access is ended immediately. Online users cannot finish their work. Their current logical units of work are rolled back (unless they are already processing a COMMIT WORK). You can change from NORMAL to QUICK. However, once the MODE is QUICK, you cannot change it back to NORMAL. When you specify QUICK for a remote server, the QUICK mode is changed to NORMAL. QUICK mode is not supported for a remote application server.
INTERVAL (positional parameter 3)	30 (seconds)	<p>The number of seconds that the CIRR transaction should delay before freeing the terminal. The value must be an integer value between 0 and 3600. This parameter controls the availability of the CICS terminal (or operator console) once you issue the CIRR transaction.</p> <p>The CICS terminal (or VSE operator console) used to activate the CIRR transaction is unavailable until the transaction ends. This could be a long time if the online application is long-running or if a user left without correctly ending the terminal session. If you issue CIRR PASSWORD,NORMAL,, server_name the terminal is not available until all online DB2 Server for VM users complete their work.</p> <p>The value you specify for interval represents an interval of time measured in seconds. If the CIRR transaction does not finish immediately, it waits the amount of time you specify. When this time ends, the CIRR transaction tries once again to finish processing. If the CIRR transaction does not finish successfully, you receive a message telling you to retry the CIRR transaction later. After issuing the message, the CIRR transaction ends. The shutdown mode is still in effect (the specified server is in the process of shutting down), and the terminal is available for your use.</p>
SERVER-NAME (positional parameter 4)	Determined by CIRB or CIRC transaction.	This parameter enables you to specify the application servers from which you want to remove access. The default server is removed if this parameter is omitted, or if the first parameter in the server_name list is blank. The default server is the one that was established by the CIRB transaction or by the CIRC transaction.

If no server_name is specified the default server_name is used. The default server_name was established by the CIRB or CIRC transaction. The CIRD transaction may be used to display the default server_name in case the user does not know what the default server_name is.

```
msg f2
AR 015 1I40I READY
2 cirr
F2-002 ARI0455I Connections to SQLMACH1 are disabled.
F2-002 ARI0460W Connections to the default server SQLMACH1 have
been disabled.
```

Figure 34. Example of CIRR with Defaults

The above example assumes that there are connections to more than one server when the CIRR transaction is entered.

If the password, mode and interval are the same then the server_name list can be used to remove connections or disable online access from multiple application servers. Since SQLVM was the last active connection, the online resource adapter was terminated. SQLMACH2 and SQLVM are local application servers, while SQLMACH8 is a remote server.

```
msg f2
AR 015 1I40I READY
2 cirr ,,,(sqlmach2,sqlmach8,sqlvm)
F2-002 ARI0455I Connections to SQLMACH2 are disabled.
F2-002 ARI0455I Online access to SQLMACH8 is disabled.
F2-002 ARI0455I Connections to SQLVM are disabled.
F2-002 ARI0413I Resource Adapter ARI00LRM is disabled.
```

Figure 35. Example of CIRR with Server-Name List

The CIRR transaction can be used to remove the connections or disable online access to the application server that were established by the CIRB and CIRA transactions. If CIRR removes the last active connections to the online resource adapter and all active APPC conversations known to the online resource adapter are deallocated, then the online resource adapter is terminated. The CIRB transaction would have to be used to restart it.

The CIRA and CIRR transactions can be entered repeatedly and in any order to add and remove links to application servers or to enable and disable online access to application servers as required.

If CIRR is entered to remove connections or disable online access to a server to which no connections or online access have been established, the message "ARI0456I Connections to <server_name> do not exist." is displayed.

If the password given on the CIRR transaction does not match the password that was used to start the connections or online access to the named server, then the connections or online access to that server are not shut down and processing continues with the next server in the list.

Displaying Information -- The CIRD Transaction

To display status information about active CICS transactions that access a local or a remote application server, issue the CICS CIRD transaction.

The CIRD transaction does not require a password, and can be issued from any CICS terminal or the operator console. To use it, you must enable it as well as the CICS restart resynchronization facility. See the *DB2 Server for VSE Installation* for more information.

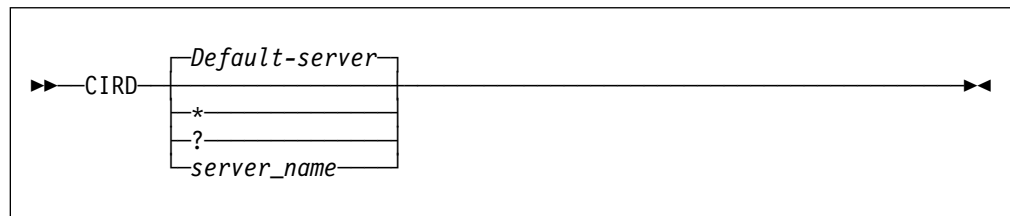


Figure 36. CIRD Transaction Syntax

The parameter is described in the following table:

Parameter	Default	Description
SERVER-NAME (positional parameter 1)	Determined by CIRB or CIRC transaction.	This parameter enables you to specify the application server whose status is to be displayed, or * to display the status of all servers and the details of transactions accessing the servers, or ? to display a list of the connected servers without the transaction details. If this parameter is omitted, the default server_name is the one that was determined by the CIRB or the CIRC transaction.

Four categories of CICS transactions access the local application server. The information that CIRD displays for transactions connected to a local server varies depending on these four categories:

- Transactions waiting to access the local application server

These transactions have issued an SQL request and are waiting because all links to the application server are busy. For these transactions, CIRD displays the elapsed time of the wait.

In general, links to the local application server are busy because other users are accessing it. The only exception occurs when the DB2 Server for VSE online support is being started; at that time, all links to the application server could be busy during the synchronization of the database log and the CICS log. Usually this requires little time, but a long delay can occur if a very large LUW is being rolled back.

- Transactions currently accessing the local application server

These transactions have established a link to the local application server and an LUW. The application server is currently doing processing for that LUW. For these transactions, CIRD displays the elapsed time of the current SQL statement, and the elapsed time the link is held. The latter effectively indicates the elapsed time of the current LUW.

- Transactions holding a link to the local application server but not using it

These transactions have established a link to the local application server and an LUW, but the application server is not currently processing for that LUW. Instead, these transactions are doing other work or are waiting for terminal communications. For these transactions, CIRD displays the elapsed time since the last application server access ended, and the elapsed time the link is held. Again, the latter effectively indicates the elapsed time of the current LUW.

- Transactions that previously held a link to the local application server, but no longer do.

These transactions have previously ended one or more LUWs, but have not yet started another. For these transactions, CIRD displays the elapsed time since the last LUW completed.

If you enter CIRD when the DB2 Server for VSE online support is not enabled or when the CIRD is not operational, an error message is displayed and CIRD ends. Note that for CIRD to display information about a transaction, the transaction must have issued an SQL request. CIRD displays the following information (where applicable) for each of the four categories of local database transactions:

- The CICS task number (TASKNO)
- The CICS transaction identifier (TRANID)
- The CICS terminal identifier (TERMID)

Not all transactions have a terminal identifier. For example, ISQL has a two-transaction structure: ISQL and CISQ. The former controls the terminal and the latter is for access to the application server. Because a CISQ transaction has no terminal associated with it, instead of displaying TERMID for it, CIRD displays the terminal identifier in another field called USERDATA (described below).

If a transaction accesses the application server, but does not have a terminal associated with it, CIRD does not display TERMID.

- The user identifier (USERID) that the application server establishes for the transaction

CIRD does not display this identifier unless a user ID has been established, which is done when an application issues an SQL statement that starts an initial LUW. The user ID may not be established immediately. (For example, a transaction can be waiting for a link to the application server.) It remains established after a transaction ends an LUW, unless the RELEASE option of COMMIT WORK or ROLLBACK WORK was used.

- User data (USERDATA) for ISQL transactions

The USERDATA field contains the terminal identifier (TERMID) of the terminal that was used to call ISQL. For most other transactions, USERDATA is blank. It is possible, however, to code an online application to initialize the USERDATA field. Such an application would use the DB2 Server for VSE online cancel support. For more information, see “Coding Your Own Cancel Exit” on page 383.

Note: If you are controlling ISQL access with the DFHSIT CMXT parameter, you have renamed the ISQL transaction. For these renamed ISQL transactions, CIRD still displays the terminal identifier of the terminal that was used to run the transaction. For more information on this parameter, see “Access Control to ISQL on a VSE Guest” on page 149.

- The elapsed time intervals (as described above)

CIRD uses the following format to display the time:

hh:mm:ss

CIRD then displays the time of day and the date, as follows:

TIME=*hh:mm:ss* DATE=*mm/dd/yy* (or *dd/mm/yy*)

and then ends its processing. (The format of the date depends on how you specified it on the DATE parameter of the VSE STDOPT JCC/JCS.)

If CIRD determines that no CICS transactions apply to the application server, it displays only the time and the date, and then ends.

Note: If the DB2 Server for VSE online support ends abnormally (for example, if the application server partition ends unexpectedly), the CIRD transaction is called implicitly to display information about transactions that were accessing the application server at the time of the failure. This information is displayed on the VSE system console.

For the following examples, assume that SQLMACH1 is the default local server and that connections have been established for the local application servers SQLMACH1, SQLMACH2 and SQLVM.

Figure 37 on page 129 shows an example of the information displayed by the CIRD transaction with no parameters.

```

2 cird
F2 002 The default server is SQLMACH1.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH1.
F2 002 Status of online DB2 Server for VSE applications:
F2 002
F2 002 Transactions waiting to establish a link to the application server are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  WAIT TIME
F2 002 -----  -----  -----  -----  -----  -----
F2 002 000033  MKE2           L222      00:01:32
F2 002 000025  INV    L224    JIM           00:08:32
F2 002
F2 002 Transactions holding a link and now accessing the application server are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  TIME USED  TOTAL LUW
F2 002                FOR CURRENT  TIME
F2 002                ACCESS
F2 002 -----  -----  -----  -----  -----  -----  -----
F2 002 000019  CISQ           DEPT222  L199     00:01:32  00:03:48
F2 002 000037  INV    L209    TERRY     00:00:01  00:00:03
F2 002
F2 002 Transactions holding a link to the application server but not using are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  TIME SINCE  TOTAL LUW
F2 002                LAST ACCESS  TIME
F2 002 -----  -----  -----  -----  -----  -----  -----
F2 002 000003  CISQ           WILLIAM  L210     00:07:01  00:10:56
F2 002
F2 002 Transactions which previously accessed the application server (not holding link):
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  TIME SINCE  TOTAL LUW
F2 002                LAST ACCESS  TIME
F2 002 -----  -----  -----  -----  -----  -----  -----
F2 002 000003  MKE2           ROBERT   L210     00:20:04
F2 002
F2 002 TIME=14:28:23 DATE=09/01/95

```

Figure 37. Example of CIRD with Defaults

Figure 38 on page 130 shows an example of the information displayed by the CIRD transaction with a server_name specified.

```

2 cird sqlmach2
F2 002 The default server is SQLMACH1.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH2.
F2 002 Status of online DB2 Server for VSE applications:
F2 002
F2 002 Transactions waiting to establish a link to the application server are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  WAIT TIME
F2 002 -----  -----  -----  -----  -----  -----
F2 002 000033  MKE2           L222      00:01:32
F2 002 000025  INV    L224    JIM           00:08:32
F2 002
F2 002 Transactions holding a link and now accessing the application server are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  TIME USED  TOTAL LUW
F2 002                FOR CURRENT  TIME
F2 002                ACCESS
F2 002 -----  -----  -----  -----  -----  -----
F2 002 000019  CISQ           DEPT222  L199      00:01:32  00:03:48
F2 002 000037  INV    L209    TERRY           00:00:01  00:00:03
F2 002
F2 002 Transactions holding a link to the application server but not using are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  TIME SINCE  TOTAL LUW
F2 002                LAST ACCESS  TIME
F2 002 -----  -----  -----  -----  -----  -----
F2 002 000003  CISQ           WILLIAM  L210      00:07:01  00:10:56
F2 002
F2 002 Transactions which previously accessed the application server (not holding link):
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  TIME SINCE  TOTAL LUW
F2 002                LAST ACCESS  TIME
F2 002 -----  -----  -----  -----  -----  -----
F2 002 000003  MKE2           ROBERT  L210      00:20:04
F2 002
F2 002 TIME=14:28:23 DATE=09/03/95

```

Figure 38. Example of CIRD with Server-Name

Figure 39 on page 131 shows an example of the information displayed by the CIRD transaction with the * specified.

```

2 cird *
F2 002 The default server is SQLMACH1.
F2 002 There are connections to server SQLMACH1.
F2 002 There are connections to server SQLMACH2.
F2 002 There are connections to server SQLVM.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH1.
F2 002 Status of online DB2 Server for VSE applications:
F2 002
F2 002 Transactions waiting to establish a link to the application server are:
F2 002
F2 002 TASKNO TRANID TERMID USER ID  USERDATA  WAIT TIME
F2 002 -----
F2 002 000033 MKE2                L222      00:01:32
F2 002 000025 INV   L224   JIM                00:08:32
F2 002
F2 002 Transactions holding a link and now accessing the application server are:
F2 002
F2 002 TASKNO TRANID TERMID USER ID  USERDATA  TIME USED      TOTAL LUW
F2 002                                     FOR CURRENT  TIME
F2 002                                     ACCESS
F2 002 -----
F2 002 000019 CISQ                DEPT222  L199      00:01:32    00:03:48
F2 002 000137 INV   L209   BOB                00:17:34    01:24:03
F2 002
F2 002 Transactions holding a link to the application server but not using are:
F2 002
F2 002 TASKNO TRANID TERMID USER ID  USERDATA  TIME SINCE     TOTAL LUW
F2 002                                     LAST ACCESS  TIME
F2 002 -----
F2 002 000013 CISQ                LARRY    L210      00:03:01    00:11:36
F2 002
F2 002 Transactions which previously accessed the application server (not holding link):
F2 002
F2 002 TASKNO TRANID TERMID USER ID  USERDATA  TIME SINCE
F2 002                                     LAST ACCESS
F2 002 -----
F2 002 000003 MKE2                LOUISA   L210      01:57:04
F2 002
F2 002 TIME=14:28:23 DATE=09/03/95
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH2.
F2 002 There are no active DB2 Server for VSE transactions.
F2 002
F2-002 TIME= 14:29:47 DATE= 09/03/95
F2 002 -----
F2 002 DBDCCICS connected to server SQLVM.
F2 002 There are no active DB2 Server for VSE transactions.
F2 002
F2 002 TIME=14:30:23 DATE=09/03/95

```

Figure 39. Example of CIRD with *

Figure 40 shows an example of the information displayed by the CIRD transaction with the ? specified.

```

2 cird ?
F2 002 The default server is SQLMACH1.
F2 002 There are connections to server SQLMACH1.
F2 002 There are connections to server SQLMACH2.
F2 002 There are connections to server SQLVM.
F2 002 -----

```

Figure 40. Example of CIRD with ?

Some extra information can be derived from the displays. In Figure 40 notice that SQLMACH1 is mentioned as the default server and on the next message that there are connections to SQLMACH1 also. It is possible, with the CIRR transaction, to

remove the connections to SQLMACH1. The CIRD command would still show that the default server is SQLMACH1 but the message indicating there are connections to SQLMACH1 would not be displayed. In this scenario, users connecting to the default server would receive SQLCODE = -940 on the CONNECT statement. The CIRA transaction could be used to establish connections to SQLMACH1 again or the CIRC transaction could be used to change the default server to one of the other active servers. Either method allows CONNECT statements to access the default server.

If CIRR or CIRT has been issued to disconnect a server or to shut down the online resource adapter but cannot complete because there are still active transactions against the server, the CIRD transaction will show which transactions and which servers are affected.

Figure 41 on page 133 shows an example of the information displayed by the CIRD transaction with the ? parameter specified. The attempt to remove the connections to SQLMACH2 fails because there are still active transactions. Then the CIRD transaction is used to determine which transactions are still active. The user is found and asked to complete his work. When the CIRR command is retried it completes successfully and the connections to SQLMACH2 are shut down.

```

2 cird ?
F2 002 The default server is SQLMACH1.
F2 002 There are connections to server SQLMACH1.
F2 002 There are connections to server SQLMACH2.
F2 002 There are connections to server SQLVM.
F2 002 -----
2 cirr ,,1,sqlmach2
F2 002 ARI0463I The DISABLE transaction CIRR must delay for a
          1-second interval before attempting the disable.
F2-002
2 cird ?
F2 002 The default server is SQLMACH1.
F2 002 There are connections to server SQLMACH1.
F2 002 Connections to SQLMACH2 are being disabled.
F2 002 There are connections to server SQLVM.
F2 002 -----
F2-002
2 cird *
F2 002 The default server is SQLMACH1.
F2 002 There are connections to server SQLMACH1.
F2 002 Connections to SQLMACH2 are being disabled.
F2 002 There are connections to server SQLVM.
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH1.
F2 002 There are no active DB2 Server for VSE transactions.
F2 002
F2 002 TIME= 19:07:43 DATE= 09/20/95
F2-002
F2 002 -----
F2 002 DBDCCICS connected to server SQLMACH2.
F2 002 Status of online DB2 Server for VSE applications:
F2 002
F2 002 Transactions holding a link to the application server but not using are:
F2 002
F2 002 TASKNO  TRANID  TERMID  USER ID  USERDATA  TIME SINCE  TOTAL LUW
F2 002                                     LAST ACCESS TIME
F2 002 _____  _____  _____  _____  _____  _____  _____
F2 002 0000129  CISQ      CICSUSER  L77D      00:00:31  00:00:31
F2 002
F2 002 TIME= 19:07:44 DATE= 09/20/95
F2 002 -----
F2 002 DBDCCICS connected to server SQLVM.
F2 002 There are no active DB2 Server for VSE transactions.
F2 002
F2 002 TIME= 19:07:45 DATE= 09/20/95
F2-002
2 cirr ,,2,sqlmach2
F2-002 ARI0455I Connections to SQLMACH2 are disabled.

```

Figure 41. Example of CIRD in a Disable Scenario

The CIRD transaction displays the following information (where applicable) for transactions that relate to a remote application server:

RDBMS

displays the name, class, and release level (version, release, and modification level) of the application server being accessed.

LU

displays the logical unit name.

|
| **TPN**
| displays the transaction program name. Its character and hexadecimal versions
| are both displayed.
|
| **TASKNO**
| displays the number of the task.
|
| **TRANID**
| displays the transaction id.
|
| **TERMINID**
| displays the name of the terminal where the transaction was initiated.
|
| **USER ID**
| displays the connected user id.
|
| **STATUS**
| displays the communication state. COMM indicates that the transaction sent an
| SQL statement to the database machine and has been waiting for a reply since
| the time shown. APPL indicates that the transaction returned control to the
| application at the time shown. VRA indicates that the Online Resource Adapter
| is processing your request. WAIT indicates that the transaction is waiting for a
| session.
|
| **TIME**
| displays the time when the STATUS displayed had begun. For example, task
| number 891 has already returned control to the application at 09:12:42, as
| indicated by TIME.
|
| **LUWID**
| displays the logical unit of work identifier, which uniquely identifies an LU6.2
| conversation. Its value is *netid.luname.instance_number.sequence_number*
| where *netid* and *luname* are up to 8 characters long, *instance_number* is 12
| characters long, and *sequence_number* is 4 characters long.

|
| Figure 42 on page 135 shows an example of the information displayed by the
| CIRD transaction with a remote server-name specified.


```

User: 2 cird sqlmach8
System: F2 0002 The default server is SQLMACH8.
F2 0002 -----
F2 0002 Status of online DB2 Server for VSE applications for
F2 0002 RDBMS = SQLMACH8 SQLDS/VM V6.1.0
F2 0002 LU = VMC3
F2 0002 TPN = SQLMACH8
F2 0002 (X'E2D8D3D4C1C3C8F8')
F2 0002
F2 0002 TASKNO  TRANID  TERMID  USER ID  STATUS  TIME
F2 0002 -----  -----  -----  -----  -----  -----
F2 0002          LUWID
F2 0002
F2 0002 0000891  DRT1    D080   SYSA      APPL    1998-08-11.09:12:42
F2 0002
F2 0002          CAIBM0ML.D08001.E31FE596ADDE.0001
F2 0002
F2 0002
F2 0002 TIME= 09:18:11 DATE= 08/11/98
F2-0002

```

Figure 42. Example of CIRD with remote server name

Figure 43 shows an example of the information displayed by the CIRD transaction with a ? specified, where online access to the remote server RMTSERV1 is allowed. Assume that SQLMACH1 is the default local application server and RMTSERV1 is a remote application server. Connections have been established for SQLMACH1 and online access to RMTSERV1 through the online support is allowed.

```

User: 2 cird ?
System: F2 0002 The default server is SQLMACH1.
F2 0002 There are connections to server SQLMACH1.
F2 0002 Online access to remote RMTSERV1 is allowed.
F2 0002 -----

```

Figure 43. Example of CIRD with ?

Stopping the Online Support -- The CIRT Transaction

While the online support is enabled, it uses CICS resources (storage) and application server resources (agents). At certain periods of the day, you may want to free these resources and prevent online access to the application server. You may, for example, want to allow only batch access to the application server for purposes of loading a large amount of data. For either of these situations, the operator can disable the online support by entering the CIRT transaction.

To end DB2 Server for VSE online support, issue the CICS CIRT transaction. The syntax of the CIRT transaction is as follows:

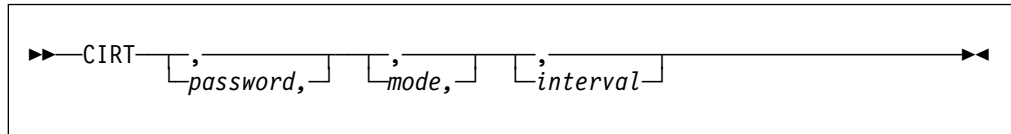


Figure 44. CIRT Transaction Syntax

Parameter	Default	Description
PASSWORD (positional parameter 1)	SQLDBAPW	This password establishes the operator's authority to terminate the online access to the application server . It must be the same password that was supplied for the CIRA or CIRB transaction. Refer to "Password Implications on Online Resource Adapter Termination" on page 140 for more details.
MODE (positional parameter 2)	NORMAL	This parameter establishes the shutdown mode: NORMAL or QUICK. When remote application servers are accessed by the online support, CIRT NORMAL will complete only when all conversations to the remote application servers are deallocated. When you specify NORMAL, the CIRT transaction prevents new online users from accessing the application server. Users who are already doing work, however, can finish. When all users complete their work, no online users can use the application server. When you specify QUICK, online access to local application servers is ended immediately. Online users accessing a local application server cannot finish their work. Their current logical units of work are rolled back (unless they are already processing a COMMIT WORK). You can change from NORMAL to QUICK. However, once the MODE is QUICK, you cannot change it back to NORMAL. When remote application servers are accessed by the online support and you specify QUICK, online access to the remote application server is <i>not</i> ended immediately. Online users accessing a remote server can finish their unit of work, but cannot start a new logical unit of work. QUICK mode is not supported for a remote application server.
INTERVAL (positional parameter 3)	30 (seconds)	<p>The number of seconds that the CIRT transaction should delay before freeing the terminal. The value must be an integer value between 0 and 3600. This parameter controls the availability of the CICS terminal (or operator console) once you issue the CIRT transaction.</p> <p>The CICS terminal (or VSE operator console) used to activate the CIRT transaction is unavailable until the transaction ends. This could be a long time if the online application is long-running or if a user left without correctly ending the terminal session. If you issue CIRT PASSWORD,NORMAL the terminal is not available until all online DB2 Server for VSE users complete their work. Even with CIRT PASSWORD, QUICK there may be some delay before the CICS terminal allows the CIRT terminal to complete its cleanup process.</p> <p>The value you specify here represents an interval of time measured in seconds. If the CIRT transaction does not finish immediately, it waits the amount of time you specify. When this time ends, the CIRT transaction tries once again to finish processing. If the CIRT transaction does not finish successfully, you receive a message telling you to retry the CIRT transaction later. After issuing the message, the CIRT transaction ends. The shutdown mode is still in effect (the specified DB2 Server for VM system is in the process of shutting down), and the terminal is available for your use.</p>

If links or online access to multiple application servers exist, they will all be removed. Once all of the links and/or online access have been removed, the online resource adapter is terminated.

The following examples assume that SQLVM, SQLMACH1 and SQLMACH2 are local application servers, and SQLMACH8 is a remote application server.

```
msg f2
AR 015 1I40I READY
2 cirt
F2-002 ARI0455I Connections to SQLVM are disabled.
F2-002 ARI0455I Connections to SQLMACH2 are disabled.
F2-002 ARI0455I Connections to SQLMACH1 are disabled.
F2-002 ARI0455I Online access to SQLMACH8 is disabled.
F2-002 ARI0413I Resource Adapter ARI00LRM is disabled.
```

Figure 45. Example of CIRT with Connections to Four Applications Servers

Note that the message ARI0413I Resource Adapter ARI00LRM is disabled is not displayed until the last application server connections and APPC conversations have been severed.

When the online resource adapter is not active, the CIRA and CIRR transactions are invalid. The online resource adapter needs to be enabled with the CIRB transaction before the CIRA and CIRR transactions can be used.

```
F2-002 ARI0413I Resource Adapter ARI00LRM is disabled.
2 cira ,,,sqlmach1
F2-002 ARI0411I Resource Adapter is not enabled.
2 cirr ,,,sqlmach1
F2-002 ARI0411I Resource Adapter is not enabled.
```

Figure 46. Example of CIRA and CIRR after CIRT

Effect of a Shutdown on Online Applications

In the NORMAL mode, CIRT prevents new LUWs from being started. As LUWs end, the links to the local application server are disconnected and APPC conversations to the remote application server are deallocated. (The NORMAL process allows for the normal end of all online LUWs.) After all links are disconnected and all APPC conversations are deallocated, the CICS storage resources are freed, and application access to the DB2 Server for VSE online support is no longer allowed.

In the QUICK mode, links to the local application server are immediately disconnected. Some online LUWs may be interrupted. The CICS storage resources are freed, and application access to online support is no longer allowed.

With QUICK, when the links are disconnected, the database manager does a ROLLBACK WORK for all LUWs that were not committed or at a synchronization point (that is, those LUWs that were prepared for COMMIT or ROLLBACK).

While the CIRT transaction is ending access in QUICK mode, the CICS transactions that access the application server can be ended by CICS with an abend code of AEY9, ASP7, or ASRA. To allow for normal transaction shutdown, then, you should either use the CIRD transaction to determine which transactions accessing the application server are still active and wait until they are complete, or use the CIRT transaction with the NORMAL option which allows all active users to finish their work.

The QUICK mode is not supported when you are ending online access to a remote server. In this case, the QUICK mode is changed to NORMAL mode.

Terminal Availability During Online Shutdown

The terminal used to activate the CIRT transaction for NORMAL or QUICK is unavailable until the transaction ends. This could be a long time for a large online application or for an online application controlled by a CICS terminal operator who is not at the console. There are two conditions when CIRT may need to wait (in CICS terms, delay for an interval of time):

- In the NORMAL mode, the process must wait until all LUWs complete normally.
- In both NORMAL and QUICK modes, after all connections and APPC conversations to the application server are severed, the process attempts to disable itself. The attempt can fail if CICS finds some online transaction that is still active and had access to the application server before CIRT was issued. In this situation, the CIRT transaction cannot complete its clean-up process until that transaction ends.

In the situations described above, CIRT will wait for an interval of time before attempting to complete the cleanup process again. (The default interval of time is 30 seconds. The interval can be specified as an input parameter to CIRT.)

After the delay, the CIRT transaction determines if the condition that caused the wait has passed. If it has, the process completes, and the online support is disabled. If not, CIRT exits by returning to CICS (the shutdown mode is still active and the terminal is free), and message ARI0414I is displayed, prompting the operator to retry the CIRT transaction later.

The operator can proceed in a number of ways to disable the online support:

- The installation may have a policy that work can continue until 5:30 PM. The operator routinely issues CIRT SQLDBAPW,NORMAL at 5 PM. Doing this prevents new work from starting. The operator then waits until 5:30 PM and reissue the CIRT transaction to proceed with normal transaction shutdown.
- The operator can use the CICS message transaction CMSG to route messages to selected terminals or users, and CICS CEMT, CIRD or CSMT commands to determine who or what applications are active, or to end the application. After such operator intervention is completed, the CIRT transaction is re-entered and the online support becomes disabled.

This intervention presupposes that the operator has information about those CICS transactions that access the application server. You may find it useful to keep a list or use a naming convention for all such transactions.

- If the NORMAL process was attempted and could not finish, the operator can escalate the shutdown mode (escalate in the sense that the database manager goes from NORMAL mode, which allows all LUWs to end, to QUICK mode, which immediately stops all access to the application server). To escalate, the operator enters CIRT SQLDBAPW,QUICK.

Shutdown Impact to Online Applications

After the online support has been disabled, or before it has been enabled, CICS abnormally ends any transaction that attempts online access to the application server by abending the transaction with abend code AEY9. If an attempt is made to execute a transaction while the online support has not been enabled, the transaction also abends with an abend code AEY9. If an application attempts to use CICS HLPI to access either a CICS/VSE subsystem or non-CICS/VSE subsystem that has not been enabled, the CICS terminal operator receives the CICS/VSE abend code AEY9.

When the shutdown process is active, the following occurs:

- For NORMAL mode, the result depends on the state of the application program. If it is in work, the process has no effect. If the application program is not in work, the online support returns an SQLCODE of -937. A later request by such a program will cause CICS to abnormally end the transaction with the AEY9 abend code.
- For QUICK mode, all initial requests result in the -937 SQLCODE, and a later request will result in the AEY9 abend code.

Also, for the QUICK mode, the online support cannot participate in the CICS two-phase syncpoint protocol. (For information on this protocol, review the discussion on online application recovery in the *DB2 Server for VM Database Administration* manual.) When the online support reports to CICS that it is disabling, the result is an ASP7 abend. This is the general abend code that the CICS syncpoint manager uses when a CICS or non-CICS/VSE subsystem cannot participate in the two-phase syncpoint protocol. Online application programs do not regain control for clean-up routines when an ASP7 abend occurs. The ISQL transaction must be ended by the operator with the CICS CSMT or CEMT command.

Password Implications on Online Resource Adapter Termination

The password used on the CIRR and CIRT transactions must be the same one that was used on the CIRA and/or the CIRB transactions. CIRR and CIRT will only shut down the connections to servers where the password matches. If the passwords do not match, that server is not shut down.

Consider the following example:

1. The online resource adapter is started with the command:
`CIRB pw1,5,,,(SQLMACH1,SQLMACH2)`
2. Connections to two new servers are added with the command:
`CIRA ,,,(SQLMACH3,SQLMACH4)`
3. Another connection is added to a fifth server with the command:
`CIRA pw2,1,,SQLMACH5`

It is not possible to end the online resource adapter with one command in this scenario. The CIRT or CIRR transactions must be run at least three times before the online resource adapter is completely shutdown because three different passwords were used to start it up.

The CIRT transaction issued with no parameters would only shut down the connections to SQLMACH3 and SQLMACH4 because they were the only servers that were started with the default password.

To shut down SQLMACH5, you would have to enter the following command:

```
CIRT pw2
```

To bring down the remaining servers and stop the online resource adapter you need to enter:

```
CIRT pw1 followed by CIRT
```

The CIRR transaction can also be used, but the server names must be specified. The following shows the CIRR commands that would be equivalent to the CIRT commands in this scenario.

```
CIRT pw1 is equivalent to CIRR pw1,,,(SQLMACH1,SQLMACH2)
```

```
CIRT is equivalent to CIRR ,,,(SQLMACH3,SQLMACH4)
```

```
CIRT pw2 is equivalent to CIRR pw2,,,SQLMACH5
```

If the command:

```
CIRR ,,,(SQLMACH1,SQLMACH2,SQLMACH3,SQLMACH4,SQLMACH5)
```

were entered only SQLMACH3 and SQLMACH4 would be disconnected.

Message ARI0464E will be issued for servers SQLMACH1, SQLMACH2 and SQLMACH5 because the passwords do not match.

Similarly, if the command:

```
CIRR pw1,,,(SQLMACH1,SQLMACH2,SQLMACH3,SQLMACH4,SQLMACH5)
```

were entered only SQLMACH1 and SQLMACH2 would be disconnected.

Message ARI0464E will be issued for servers SQLMACH3, SQLMACH4 and SQLMACH5 because the passwords don't match.

Chapter 6. Maintaining Database Security

This chapter discusses the methods available for protecting the information in a database:

- Communications and system security

The security mechanisms to restrict unauthorized access to the database:

- Session-level security

The VTAM product supports a security exchange between two partner logical units (LUs) that are attempting to establish a session with each other. This security exchange is called *partner LU verification*.

- Conversation-level security

The APPC/VM communications allow an application requester to specify one of the following levels of conversation security when it is trying to connect to the application server:

- SAME
- PGM

- VM directory control statements

Use these statements to control the ability of an application requester to communicate with the application server.

- User ID translation

Because the application server cannot differentiate between identical user IDs from local and remote systems, the ability to translate inbound user IDs to locally known ones can eliminate duplicate user IDs and avoid potential security-related problems.

- Minidisk protection

Use the guidelines in this section to protect your database minidisks against unauthorized access.

- CMS restrictions

Certain CMS file manipulation commands must not be used on database files because they can render the database useless.

- System and DB2 Server for VM operator console considerations

The integrity and security of your database can be threatened if unauthorized persons have access to the console. This section discusses how to use the CP DISCONN command to restrict console access.

- Access control to ISQL on a VSE guest

If VSE guest users use ISQL to access an application server on a VM/ESA operating system, you may want to limit their access.

Communications and System Security

This section discusses the following topics:

- Security at session and conversation levels
- VM directory control statements that control communications between application requesters and application servers
- User ID translation
- Protection of database minidisks.

Session-Level Security

Session-level security is controlled by the security acceptance (SECACPT) and VERIFY parameters, which are used when an LU is defined in the VTAM product. The type of session-level security that is specified determines the type of conversation-level security that is supported. If SECACPT is set to CONV, only PGM can be specified at the conversation level; if it is set to ALREADYV, both PGM and SAME can be specified. In addition, partner LU verification can be specified when conversation-level security is SAME (that is, SECACPT=ALREADYV).

Note: You cannot specify SECACPT=ALREADYV if you are using VTAM Version 3 Release 2.

Partner LU verification is recommended for connections that the VTAM product routes through an AVS gateway and that specify SECURITY=SAME. These connections do not provide a user ID and password to be validated at the target site. (When partner LU verification occurs, the identity of each SNA LU is confirmed.) For more information on partner LU verification, see the *Distributed Relational Database Connectivity Guide*, and the *VTAM Resource Definition Reference* manuals.

Notes:

1. In VM, an LU is also known as an AVS gateway.
2. If a connection is routed to an application server through AVS with SECURITY=SAME, then AVS user ID translation must be used. For more information, see "User ID Translation" on page 147.

Conversation-Level Security

Conversation-level security can be specified by both DB2 Server for VM and non-DB2 Server for VM users connecting to a DB2 Server for VM application server. Non-DB2 Server for VM users are remote users by definition, and are routed to the server by the VTAM product through an AVS gateway. The DB2 Server for VM users, on the other hand, can access a DB2 Server for VM application server on the local system, and through an AVS gateway or TSAF collection. In all situations, the user can specify the level of conversation security. The type of conversation-level security that is supported depends on the session-level security that is specified for the underlying system. The DB2 Server for VM application server accepts a conversation-level security of either SAME or PGM, and either rejects or accepts the connection on that basis. The default used by the DB2 Server for VM application requester for the user is SAME.

Connections that specify SECURITY=SAME do not provide any security identification. Only a user ID is provided, which is validated by the source host by using CP, RACF, or an equivalent security manager product. Because the user ID is validated by the source host, it is considered to be already verified by the target host. The user ID that is sent is **always** the VM user ID. The value that is specified for the `:userid` tag in the CMS communications directory is not used for SECURITY=SAME connections. No password is sent.

The AGW ADD USERID command must be issued at the target AVS machine to authorize inbound SECURITY=SAME connections. If this command is not issued, these connections are rejected; if it has been used to map an inbound user ID at the target AVS machine, the target host does **not** validate that user ID. The connection is accepted, whether or not the mapped user ID exists on the target host. If the command

```
AGW ADD USERID remotelu * =
```

is issued, then the AVS machine will accept all already verified user IDs coming from `remotelu`. However, the command

```
AGW DELETE USERID remotelu remuser
```

cannot be used to delete `remuser` from the AVS user ID table because the table does not contain an entry for the `remotelu/remuser` pair. Instead, use

```
AGW DELETE USERID remotelu *
```

to delete all mappings corresponding to the remote LU `remotelu`.

Note: More than one user will be affected when `AGW DELETE USERID remotelu *` is issued.

If the command **AGW ADD USERID remotelu=** is issued, then the command **AGW DELETE USERID remotelu userid** is issued, `userid` is **not** deleted from the AVS user ID table. The AVS machine will accept all user IDs coming from `remotelu`, including `userid`. To undo the AGW ADD USERID command, issue the AGW DELETE USERID command.

If the VTAM product is routing a connection through an AVS gateway, the AVS translation feature must be used to map an inbound user ID to a user ID that is locally defined. The mapped user ID is validated by the installed security manager product such as RACF or CP. A mapping is enabled and disabled with the AGW ADD USERID and AGW DELETE USERID commands.

Note: The AVS translation feature requires VTAM Version 3 Release 3 or later. You can only use this feature when SECURITY=SAME.

Connections that are specified as SECURITY=PGM provide a user ID and a password to be validated by the installed security manager product (for example, CP, RACF, or an equivalent product). If the validation is successful, the connection is accepted; otherwise, it is rejected.

For more information about conversation-level security, see the *Distributed Relational Database Connectivity Guide*, and the *VM/ESA: Connectivity Planning, Administration, and Operation* manuals.

Note: Conversation-level security is supported by APPC/VM but not by IUCV.

VM Directory Control Statements

The VM operating system provides control statements that can be added to the directory entry of any virtual machine, both to enable functions and to restrict access on a particular processor.

Access to the database can be controlled by controlling access to the application server that manages it. Communications between an application requester and an application server can be enabled selectively with VM control statements that are added to the directory entries of one or both of the virtual machines. (The application server virtual machine is also referred to as the database machine.)

You can use either the user machine or the database machine to control access authority. If you want to allow all virtual machines on the same processor to connect to the application server, add the IUCV ALLOW control statement to the database machine directory entry. If you want to limit access to a particular application server to a small group of users, add the IUCV *resid* control statement to the directory entry of each user machine requiring access, and leave out the IUCV ALLOW control statement in the database machine directory entry.

Control Statements for VM/ESA Environments

In the VM/ESA operating systems, you can use the control statements IUCV ALLOW, IUCV ANY, and IUCV *IDENT to control the access that application requesters have to application servers.

IUCV ALLOW: When this statement is added to the directory entry of the database machine, all application requesters and communications servers (such as TSAF and AVS) on the same processor can connect to the application server.

IUCV ANY: When this statement is added to the directory entry of a virtual machine, the application requester can connect to all application servers and communications servers (such as TSAF and AVS) that are on the same processor. To limit access to specific users, use this statement in each user machine requiring access, instead of specifying IUCV ALLOW in the database machine.

You can further limit access to specific application servers by adding one or more IUCV *resid* statements to the directory entry.

Attention: If the VSE guest user machine directory entry contains the IUCV ANY statement, then anyone who knows the CIRB transaction password has database administrator (DBA) authority on all application servers.

IUCV *IDENT: All databases are identified as VM resources with either a local or global scope. A local resource can be accessed only by an application requester residing on the same processor. A global resource can be accessed by an application requester that is either local or remote.

The virtual machine where the application server resides (the database machine) uses the IUCV *IDENT control statement to identify which resources it manages, and whether the resources are local or global. For example, the statement IUCV *IDENT SALESDB GLOBAL identifies a global resource (database) named SALESDB, whereas the statement IUCV *IDENT SALESDB LOCAL identifies the same database as a local resource.

Distributed Processing Security

The database manager takes full advantage of the TSAF and AVS communications servers to make the application server accessible to both local and remote users. The latter can be either DB2 Server for VM or non-DB2 Server for VM users.

Neither AVS nor TSAF depends on the selected protocol (either SQLDS or DRDA), nor does the selected protocol depend on either AVS or TSAF. The contents of the data streams that the communications servers facilitate are independent of both TSAF and AVS.

If you want to use TSAF and AVS to route communications, you must do the required setup tasks, including the addition of the VM directory control statements, described above. In addition, you must define your database as a global resource if you intend to support distributed processing.

For more information about VM directory control statements, see the *VM/ESA: Connectivity Planning, Administration, and Operation* manual.

Distributed Processing Administration

In a distributed environment, the role of the system administrator in authorizing and activating new user IDs, resource identifiers, and AVS gateway names is very important. You must enforce the following rules as strictly as possible:

- User IDs must be unique on the processor.
- Resource identifiers must be unique within the scope of an AVS gateway and a TSAF collection.
- All AVS gateway names must be unique within the host SNA network.
- Database names must be unique within the scope of all the SNA networks interconnected with the host SNA network.

Note: Resource identifiers and AVS gateway names must not be the same as a user ID (other than that of the resource manager), and they must not be specified as ALLOW, ANY, or SYSTEM. Also, an AVS gateway name must not be the same as any resource identifier.

For more information on VM directory control statements and on setting up TSAF and AVS virtual machines, the see *VM/ESA: Connectivity Planning, Administration, and Operation* manual.

User ID Translation

Each VM user ID is unique on a processor. However, in an environment of SNA networks and TSAF collections, in which many processors can be interconnected and user IDs are passed around for validation, the issues of duplication and ambiguity must be considered.

The application server cannot identify whether a user ID forwarded by the application requester belongs to a local or a remote user. For example, suppose that user ID STEVE is on a remote system that can access the application server through the AVS gateway; and that the same user ID exists locally and has a high level of authority. The application server cannot differentiate between the local and the remote users and treats them equally. This situation poses a risk to security, and must be eliminated.

The security risk is maximized if the VTAM product routes the remote user request through an AVS gateway, the remote user specifies SECURITY=SAME, and partner LU verification is not performed. In this situation, you should use AVS to translate the inbound user ID to one that is registered with the local security manager product. The translated user ID is validated by the local security manager product, which can be CP, RACF, or an equivalent product.

The situation described cannot happen if the remote user is using SECURITY=PGM. This user must obtain a user ID and password from the local system administrator **before** being able to specify SECURITY=PGM to access an application server.

Note: Do not allow remote users to use a user ID that already exists on the local system and that has been assigned to a local user.

Minidisk Protection

To help prevent unauthorized or malicious access to the database minidisks, do the following:

1. Code both a read-sharing password and a write-sharing password on the MDISK control statement for each minidisk.
2. Specify a read (R) access mode on the MDISK control statement for each minidisk to prevent more than one application server (or single user mode user) from accessing the database minidisks at the same time.
3. Carefully control the set of users who know the minidisk passwords, and ensure that they properly protect these passwords.

CMS Restrictions

The CMS RESERVE command creates a CMS-like file for each database minidisk. If you were to issue a CMS LISTFILE command for the database minidisks, you would, in fact, see that a file resides on each minidisk. These files, however, are not the same as regular CMS files.

Attention: Never use the following CMS file manipulation commands or macros on the database files, because they can render the database useless:

- ACCESS (with the ERASE option)
- ERASE
- EXECIO
- FORMAT (the minidisk after the database is created)
- MOVEFILE (to write to the file)
- RESERVE (the minidisk after the database is created)
- READCARD
- DISK LOAD
- RECEIVE (with the REPLACE option)
- FSERASE
- FSWRITE.

System and DB2 Server for VM Operator Console Considerations

The system console should never be left unattended. To protect the integrity of your database, always have the operator sign off the console before leaving, by entering:

```
#CP SET RUN ON
#CP DISCONN
```

This command enables the operator to sign off without stopping the application server. To restrict access to the DB2 Server for VM operator console, only give the password for the application server to people who need to use it, and who can be trusted not to misuse it or to give it to other.

Access Control to ISQL on a VSE Guest

In VSE, ISQL is made up of two transactions: ISQL and CISQ. The former controls the CICS terminal, and the latter controls access to the application server. By creating the second transaction dynamically (instead of hard-coding it as CISQ) you can put different departments or different groups of users into different CICS classes. Each group would have different transaction identifiers for both transactions of ISQL. Because the different groups have different CICS classes, you can limit the number of active ISQL users in each group.

To implement this, create any transaction ID for the first transaction. Then, instead of making CISQ the second transaction ID, make it identical to the first one except for the last character, which should be a 2. For example, if there are five departments, you could have chosen these transaction IDs:

First Transaction ID	Second Transaction ID	Department
-----	-----	-----
ISQL	ISQ2	202
ACCT	ACC2	ACCOUNTING
SAL	SA2	SALES
IN	I2	INVENTORY
P	P2	PLANNING

These examples show how the format works for different identifier lengths. Note that when the first transaction ID is one character (P), the 2 is added (P2). Also note that the first transaction ID cannot end with a 2.

Next, decide what the maximum number of ISQL users for each department should be:

First Transaction ID	Second Transaction ID	Department	Maximum ISQL Users
-----	-----	-----	-----
ISQL	ISQ2	202	2
ACCT	ACC2	ACCOUNTING	3
SAL	SA2	SALES	4
IN	I2	INVENTORY	3
P	P2	PLANNING	2

Next, specify the CICS parameters TRANSID, TCLASS, and CMXT as follows:

- TRANSACTION parameter in the CICS System Definition File

You must code an entry for each transaction ID defined. In the above example these are: ISQL, ISQ2, ACCT, ACC2, SAL, SA2, IN, I2, P, and P2. The TRANSACTION must specify the particular transaction ID (for example, TRANSID=ISQ2 for the ISQ2 transaction), and the program name parameter should reference the same program as CISQ or ISQL.

- TCLASS parameter and CMXT parameter in the DFHSIT

To fully understand these two parameters, it is best to consider them together. To implement the above example, you would code them as follows:

```

DEFINE TRANSACTION(ISQL) GROUP(DB2610) PROGRAM(ARIITRM)          *
      TWASIZE(300) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES)

DEFINE TRANSACTION(ISQ2) GROUP(DB2610) PROGRAM(ARIISQL)          *
      TWASIZE(0) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES) TCLASS(1)

DEFINE TRANSACTION(ACCT) GROUP(DB2610) PROGRAM(ARIITRM)          *
      TWASIZE(300) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES)

DEFINE TRANSACTION(ACC2) GROUP(DB2610) PROGRAM(ARIISQL)          *
      TWASIZE(0) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES) TCLASS(2)

DEFINE TRANSACTION(SAL) GROUP(DB2610) PROGRAM(ARIITRM)          *
      TWASIZE(300) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES)

DEFINE TRANSACTION(SA2) GROUP(DB2610) PROGRAM(ARIISQL)          *
      TWASIZE(0) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES) TCLASS(3)

DEFINE TRANSACTION(IN) GROUP(DB2610) PROGRAM(ARIITRM)           *
      TWASIZE(300) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES)

DEFINE TRANSACTION(I2) GROUP(DB2610) PROGRAM(ARIISQL)           *
      TWASIZE(0) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES) TCLASS(4)

DEFINE TRANSACTION(P) GROUP(DB2610) PROGRAM(ARIITRM)            *
      TWASIZE(300) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES)

DEFINE TRANSACTION(P2) GROUP(DB2610) PROGRAM(ARIISQL)           *
      TWASIZE(0) INDOUBT(BACKOUT) SPURGE(NO) TPURGE(YES) TCLASS(5)

```

These TCLASS values correspond to the positional values in the CMXT parameter. These values are arbitrary, but you should set them up so that a transaction's TCLASS value corresponds to its CMXT positional parameter. In the above example, ISQ2 has a TCLASS value of 1. This means that it is in class 1, which corresponds to the first positional value on the CMXT parameter. The first positional parameter value for CMXT is 2. This means that the maximum number of transactions that can be active in class 1 (TCLASS=1) is 2. Therefore, the number of active Department 202 users of the ISQL-ISQ2 transactions is limited to 2. The same is true for the other TCLASS and CMXT positional values. (For unspecified CMXT values, the default is 1.)

Chapter 7. Managing Database Storage

This chapter discusses:

- Database storage concepts
- Adding dbspaces to a database
- Expanding the page tables in the directory
- Acquiring dbspaces for packages
- Managing storage pools.

Storage Concepts

A database contains user data objects (tables and indexes), and supporting information maintained by the database manager. Specifically, it contains:

- A *directory* that is a minidisk containing database control information, including mappings of the dbspaces to their addresses on DASD. The directory relates the logical database image to the physical storage used.
- Either one or two *log minidisks* which hold records that describe each change made to the database. If any changes made to the data must be undone or redone, logs can be used to restore the data to a consistent state.
- One or more *storage pools*, which are collections of minidisks. These minidisks are called database extents. This is where the actual data is stored.

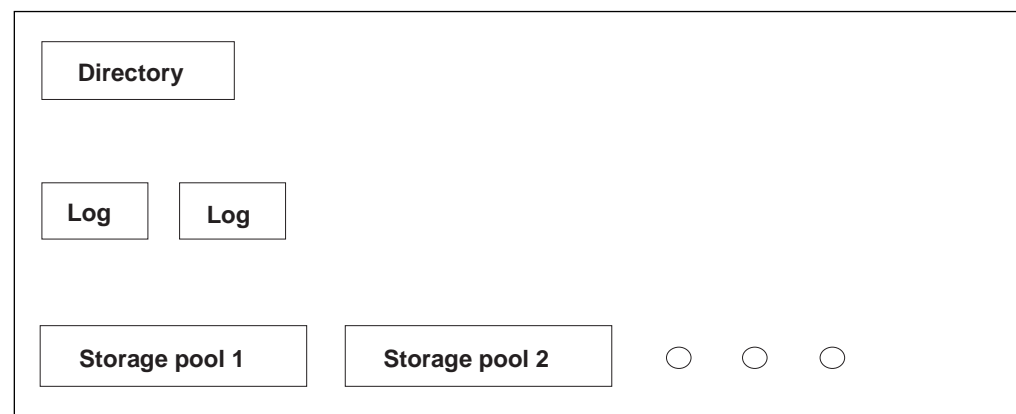


Figure 47. The DB2 Server for VM Database

A *dbextent* is an allocation of actual DASD space. *Storage pools* are composed of one or more *dbextents*. The size of a storage pool can be increased by adding more *dbextents*, or reduced by deleting existing ones. Each *dbextent* is a CMS minidisk. When dbspaces are assigned to a storage pool and their pages are filled, physical DASD pages are taken from the *dbextents* of the storage pool.

Storage pools can be defined so that they are either recoverable or nonrecoverable. By default, storage pools are recoverable, that is, the database manager does full recovery for them. For nonrecoverable storage pools, only limited recovery is done. For more information on nonrecoverable storage pools, refer to "Nonrecoverable Storage Pools" on page 250.

A *dbspace* is a logical allocation of space in the database, divided into 4096-byte blocks called pages. A *dbspace* is not a real allocation of DASD space, but only an

allocation of page tables in the directory. These page tables map logical dbspace pages to DASD locations. The database manager dynamically allocates real DASD storage space to support dbspace pages on a demand basis so unused pages do not occupy DASD space.

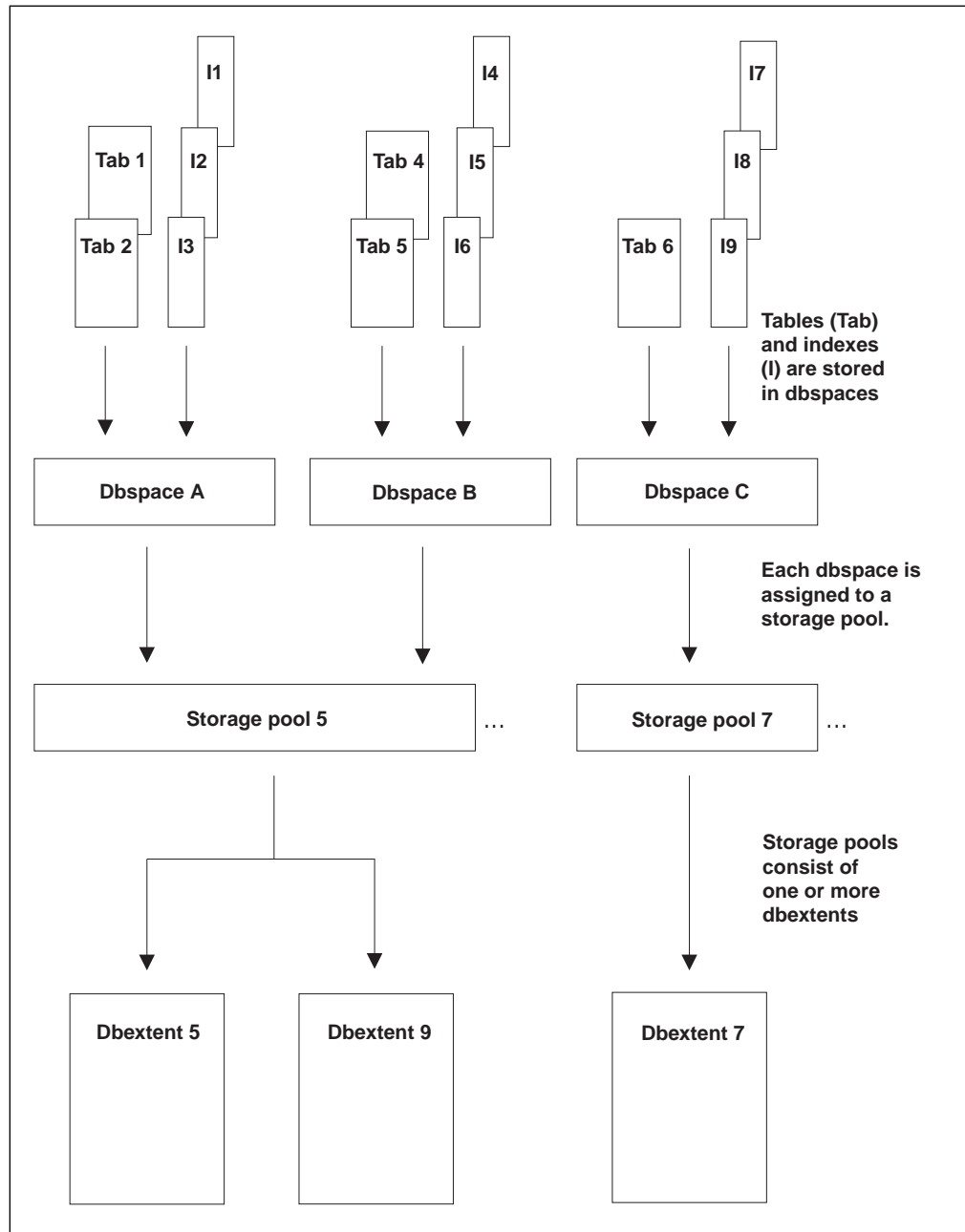


Figure 48. Physical Database Concepts

How Information is Stored in Dbspaces

Tables and their indexes are stored in dbspaces. At the beginning of every dbspace are one to eight *header pages*, which contain control information on the tables and indexes that follow. Next come data pages, which hold the rows of the tables. At the end are index pages, which hold the index entries. A page in a dbspace is

defined as a header page, a data page, or an index page, when the dbspace is acquired. Figure 49 on page 153 shows how information is stored in a dbspace.

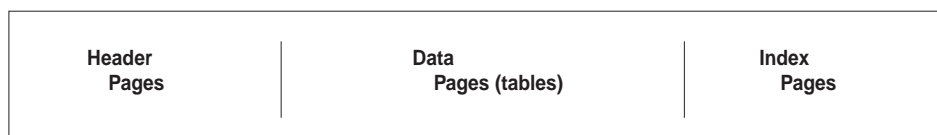


Figure 49. Table and Index Storage in a Dbspace

When a table is created, its creator can either assign it to a dbspace explicitly by specifying a dbspace in the CREATE TABLE statement, or can let the database manager assign it to a default dbspace. Any indexes created on the table obtain their storage from the same dbspace as that table.

Figure 48 shows two tables and their indexes in dbspace A, two tables and their indexes in dbspace B, and one table with three indexes in dbspace C.

The potential capacity of a dbspace is fixed when it is defined with the ADD DBSPACE command. A dbspace can hold up to 255 tables along with their indexes.

More than one table can be stored in the same dbspace, but a table cannot reside in multiple dbspaces. If you store multiple tables in a dbspace, be aware that the database manager may store rows from different tables on the same data pages. For performance reasons, it is frequently desirable to have only one table per dbspace. (Index entries from different indexes are never stored on the same page.)

There are three types of dbspaces: private, public, and internal. For private data, there should be one *private* dbspace reserved for each user. These are locked at the dbspace level, so the database manager does not incur unnecessary overhead while users are accessing their own private data. Any tables that are to be accessed by multiple users who will be doing UPDATE, INSERT, or DELETE operations should be placed in *public* dbspaces, which have page- or row-level locking to support concurrent access. *Internal* dbspaces are temporary spaces used only by the database manager to perform tasks such as sorting.

Adding Dbspaces to the Database

Before tables and indexes can be stored in a dbspace, the dbspace must be *added*, and then *acquired*. Adding a dbspace to a database consists of reserving page tables in the directory, assigning the dbspace to a storage pool, and specifying it as public or private. You can add dbspaces to the database using the SQLADBSP EXEC. This EXEC resides on the service disk (V-disk) and can be run only by a database machine in single user mode. For this reason, you should add enough dbspaces for your future needs.

You cannot remove a dbspace after it has been added to the database. After it has been acquired, you can drop its contents with the DROP DBSPACE operation so that another user can acquire it.

Figure 50 on page 154 shows the format of the SQLADBSP EXEC.

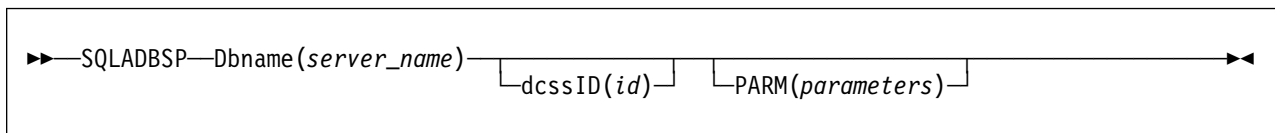


Figure 50. SQLADBSP EXEC

Dbname(*server_name*)

The DBNAME parameter is required. Any initial substring for DBNAME can be used as the keyword (for example, DB or D). For *server_name*, specify the name of the application server. (The name of the application server is defined when the SQLDBINS EXEC is started to generate the database.)

dcssID(*id*)

The DCSSID parameter is optional. You can use DCSSID or ID for the keyword. For *id*, you can specify the name of the bootstrap package that identifies saved segments. If not specified, DCSSID defaults to SQLDBA, which results in the SQLDBA bootstrap package being used, and the database manager using the default saved segments. If you do not have default saved segments, DB2 Server for VM code is loaded into the user free storage area.

PARM(*parameters*)

The PARM parameter is optional. Use it to specify additional initialization parameters. Usually, the initialization parameters used by the SQLADBSP EXEC are sufficient. You can specify other initialization parameters, as required.

Note: If you have been accessing the database with archiving, you must specify LOGMODE=A or L, as appropriate.

If you specify the PARM parameter, it must follow the other SQLADBSP parameters. For a list of the valid initialization parameters, refer to Figure 11 on page 78. That figure lists the parameters that apply in single user mode. Do not specify the SYSMODE and STARTUP parameters. The SQLADBSP EXEC automatically supplies SYSMODE=S and STARTUP=S. Also, do not specify the PROGNAME and DUALLOG parameters. The PROGNAME parameter is not valid with STARTUP=S, and the DUALLOG parameter does not apply.

You can specify the DUMPTYPE, TRACDBSS, TRACRDS, TRACDSC and TRACCONV parameters. For more information, see “Multiple User Mode Initialization Parameters” on page 58. The ADD DBSPACE operation requires that the database manager be run in single user mode, therefore, the TRACDBSS, TRACRDS, TRACCONV and TRACDSC initialization parameters are the only means of doing a trace of the ADD DBSPACE operation. (Operator TRACE commands can only be entered when the database manager runs in multiple user mode.)

If you use tracing, consider issuing your own CMS FILEDEF and LABELDEF commands for the trace file. For more information on the FILEDEF and LABELDEF commands, see the *DB2 Server for VM Diagnosis Guide and Reference* manual. For more general information about tape capabilities, see “Tape Support” on page 79.

You can use PARMID to specify a CMS file that contains parameter specifications for the ADD DBSPACE operation.

When running, the SQLADBSP EXEC must have information about the dbspaces to be added to the database. The SQLADBSP EXEC gets this information in either of these ways:

1. You create a CMS file called *resid* SQLADBSP on the database machine A-disk before running the EXEC. The *resid* is the VM resource ID associated with the application server. In a VM/ESA operating system, *resid* and server name may be different. The RESID NAMES file on the production disk is used to map the *resid* to the server name defined during database generation.
2. The SQLADBSP EXEC creates the *resid* SQLADBSP file for you while it is running. For an example of this file, see Figure 51. If the file *resid* SQLADBSP exists on the A-disk, the SQLADBSP EXEC prompts you whether to use the file or erase it.

If you choose to have the file created dynamically, the SQLADBSP EXEC prompts you to enter the number of public, private, and internal dbspaces to be added to the database. You are then prompted for the number of pages and storage pool assignments for each.

For more information on using the SQLADBSP EXEC to add dbspaces, see “Example of Adding a Dspace to a Database” on page 158.

In either situation, the SQLADBSP EXEC gives you the option of editing the *resid* SQLADBSP file with the CMS XEDIT facility. For example, if you wish to decrease the number of internal dbspaces, you will have to edit the file to change the number.

```
PUBLIC 1024 7
PUBLIC 1024 8
PRIVATE 256 5
PRIVATE 256 5
PRIVATE 256 5
PRIVATE 256 5
INTERNAL 50 1024 9
```

Figure 51. Example of a *resid* SQLADBSP File

Notice that the format for the ADD DBSPACE control statements is the same as the format for defining them during database generation.

When the *resid* SQLADBSP file is created and (optionally) edited, the SQLADBSP EXEC starts the application server in single user mode with the ADD DBSPACE option. The ADD DBSPACE operation uses the control statements in the *resid* SQLADBSP file.

Specify each dspace to be added as a record in the *resid* SQLADBSP file that contains the type (public or private), the size (number of pages), and, optionally, the storage pool assignment. (The default storage pool number is 1.) The number you specify for the size should be a multiple of 128, since directory page tables are allocated in multiples of 128-page table entries. If it is not, the database manager rounds it up to the next higher multiple of 128. Separate all parameter values by at least one blank.

On the last dbospace specification record you must specify the internal dbospaces to be defined. This record contains the keyword INTERNAL, the number of internal dbospaces to be supported, the size of each (in number of pages), and, optionally, the storage pool assignments. Internal dbospaces can be assigned to either recoverable or nonrecoverable storage pools. However, for performance reasons, the internal dbospaces should not be assigned to storage pool 1 and preferably should be stored in their own storage pool.

It is necessary that you respecify the internal dbospace values each time you add a new public or private dbospace, even if you are not changing these values from what they were before. The internal dbospace specification overrides the previous one, including changing the storage pool assignment.

Note: You may sometimes want to change the internal dbospace specifications for reasons other than adding new user dbospaces. To do this, run the SQLADBSP EXEC. When you are prompted to enter the number of public and private dbospaces, respond “0” to these. When you are prompted to enter the number of internal dbospaces to be added, enter a value. The number you specify is added to the number of internal dbospaces with which the database was generated. You are then prompted for the number of pages and storage pool assignment; enter these. Finally, you are asked if you want to modify the SQLADBSP file; respond “1” (for yes). You are now given the opportunity to change the number of internal dbospaces. This allows you, for example, to decrease the number of internal dbospaces in the database.

For example, suppose your database was generated with 50 internal dbospaces, you want to add 4 dbospaces, and you want 4096 for the number of pages. When you run SQLADBSP, you receive a message saying that the database was generated with 50 internal dbospaces. Then you are asked to enter the number of internal dbospaces you want to add and the number of pages for each dbospace. Specify 4 and 4096, respectively. If you check the SQLADBSP control file, the INTERNAL statement shows that there are 54 internal dbospaces having 4096 pages.

Now, suppose you rerun SQLADBSP. Again, you receive a message saying that the database was generated with 50 internal dbospaces, and you are asked to enter the number of internal dbospaces you want to add and the number of pages for each dbospace. Specify 2 for the number of dbospaces you are adding and 1024 for the number of pages. If you now check the SQLADBSP control file, the INTERNAL statement shows that there are 52 internal dbospaces each having 1024 pages.

Considerations for Adding Dbospaces

The ADD DBSPACE operation updates the directory and the catalog tables in the database. Only the updates to the catalog tables are recorded in the log; updates to the directory are not. Because of this, you can have a problem if you normally archive the database, and then try to restore it. Suppose the following events occur:

1. You do a database archive.
2. Later, you add dbospaces.
3. Later, users acquire and use those dbospaces.

4. You do an archive restore using the archive file that you created in step 1 and, if you use LOGMODE=L, the subsequent log archives.

The directory and the database are not synchronized. The directory has been restored from a database archive file that does not reflect the ADD DBSPACE operation. The database is also restored from that file; but its restore includes the updates recorded in the log or log archives, which do reflect the ADD DBSPACE operation. Thus, the directory does not include the new dbspaces but the database does.

To prevent this problem, archive the database immediately after the ADD DBSPACE operation, as follows:

1. After you add the dbspaces, warm-start the application server in multiple user mode (SYSMODE=M) with LOGMODE set to L or A.
2. Immediately take a new database archive, with either the ARCHIVE, SQLEND ARCHIVE, or SQLEND UARCHIVE command. (If you use SQLEND UARCHIVE, remember to take the user archive after the application server ends.)

Following this procedure will ensure that your current database archive reflects the added dbspaces. (See “Archiving Procedures” on page 211 and “Restoring the Database” on page 224 for more information on archiving and restoring procedures.)

If you do log archiving and restore the database using a database archive taken before the ADD DBSPACE operation, the same problem that was described above occurs. If you use a back-level database archive and subsequent log archives to restore the database, the database archive that records the addition of the dbspaces is skipped: the directory is restored from the back-level database archive and does not show the addition of the dbspaces, but the subsequent log archives do.

If you used the ADD DBSPACE operation only to reconfigure your internal dbspaces, restoring a back-level database does not unsynchronize the directory and database, since information about internal dbspaces is stored in the directory but their use is not recorded in the database. Thus, if you restore a back-level database, the number and size of the internal dbspaces return to the back-level values.

The ADD DBSPACE operation is a two-phase process. The first phase updates the database directory with the information about the new dbspace. The second updates the SYSTEM.SYSDBSPACES catalog table.

Completion of the first phase is indicated by the message:

```
ARI0915I  DBSPACE ADDED TO DATABASE
```

If an abnormal end occurs before message ARI0915I is issued, restart the ADD DBSPACE operation from the beginning by rerunning the SQLADBSP EXEC. If an abnormal end occurs after message ARI0915I is issued, restart the ADD DBSPACE operation by doing a start up of the application server as follows:

```
SQLSTART DB(server-name) PARM(SYSMODE=S,STARTUP=W,PROGNAME=ARISEGB)
```

Example of Adding a Dbspace to a Database

When you create a new storage pool, you must also assign at least one dbspace to the new pool to make it usable. Assigning dbspaces requires that you run the SQLADBSP EXEC. Figure 52 shows the procedure to add one public dbspace to pool number 2 in the database named TEST. Note that the example indicates the entries you make.

```
1  → sqladbsp db(test)
ARI0717I Start SQLADBSP EXEC: 01/19/93 21:11:43 EST.
ARI0648A Enter the number of PUBLIC DBSPACES to add to the database.
2  → 1
ARI0649A Enter the number of pages and storage pool assignment
      for 1ST PUBLIC DBSPACE.
3  → 512 2
ARI0648A Enter the number of PRIVATE DBSPACES to add to the database.
4  → 0
ARI0603I The database was generated with
      80 internal DBSPACES.
ARI0648A Enter the number of internal DBSPACES to add to the database.
5  → 0
ARI0638D Do you want to modify the TEST SQLADBSP file?
      Enter 0(No) or 1(Yes).
6  → 0
```

Figure 52. SQLADBSP Example of Adding a Dbspace

Notes for Figure 52:

- 1** Command to begin the ADD DBSPACE operation. Because no parameters are specified, dcssid defaults to SQLDBA, POOL defaults to LOG, and PARM defaults to the values shown in Figure 11 on page 78.
- 2** 1 is entered to add one public dbspace.
- 3** This entry specifies the size and storage pool location of the public dbspace.
- 4** 0 is entered. No private dbspaces are added.
- 5** No additional internal dbspaces will be added, so 0 is entered.
- 6** Because nothing needs changing, enter 0 (No).

After **6**, the SQLSTART EXEC is automatically called. When this EXEC ends, the SQLADBSP EXEC also ends, and the dbspace has been added.

To confirm that a dbspace has been added to the new storage pool, restart the application server use either the DBS utility or ISQL to issue this query:

```
SELECT DBSPACENO, DBSPACENAME, -
      POOL FROM SYSTEM.SYSDBSPACES -
      ORDER BY DBSPACENO
```

This query produces a table showing the dbspace numbers, dbspace names, and the number of the pool each dbspace is assigned to.

Expanding the Database Directory

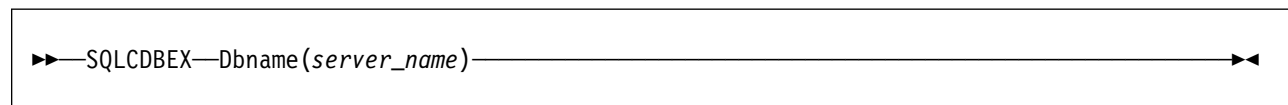
When a database is initially generated, a calculation is made to determine which portion of the directory will be set aside for the page map table, and which portion will be used for the allocation bitmaps. The size of the page map table determines the maximum number of DBSPACE pages, that is, the maximum logical size of the database. The size of the allocation bitmap determines the maximum number of dbextent pages, that is, the maximum physical size of the database. As the database grows in size with use, it may run short on either logical or physical space. If it is short on logical space, the ADD DBSPACE operation may fail. If it is short on physical space, the ADD DBEXTENT operation may fail. You can expand the directory to correct these situations.

Prior to SQL/DS Version 3 Release 5, the directory could be expanded to accommodate more logical pages by expanding the page map table. With SQL/DS Version 3 Release 5 and later, you can increase the size of the allocation bitmaps and the page map table concurrently. If it is necessary to expand the directory to hold more dbextent pages, the only available option would be to expand the directory for both dbspace and dbextent pages.

Use SQLCDBEX to increase:

- The maximum number of dbspace pages, by expanding the page map table.
- The maximum number of dbspace pages and dbextent pages, by expanding the page map table and allocation bitmaps concurrently.

The format is:



```
▶▶—SQLCDBEX—Dbname(server_name)————▶▶
```

Figure 53. SQLCDBEX EXEC

Dbname(*server_name*)

The DBNAME parameter is required. Any initial substring for DBNAME can be used as the keyword (for example, DB or D). For *server_name*, specify the name of the application server. (The name of the application server is defined when the SQLDBINS EXEC is started to generate the database.)

Figure 54 on page 160 is an example of using SQLCDBEX to expand the directory to hold more dbspace and dbextent pages.

```

sqlcldbex dbname(sqlvm350)
ARI0717I Start SQLCDBEX EXEC: 09/28/95 16:41:50 CET.
ARI0721I Get DB2 Server for VM production minidisk WRITE access: SQLDB350 195.
DASD 0195 LINKED R/W; R/O BY    2 USERS

ARI6102A Enter DBEXTENT number (or LOGDSK1, LOGDSK2,
          or BDISK) to copy.
          (Enter a null response to end input or
          enter QUIT to exit.)

bdisk
ARI6188A Enter the output block size of the directory.
          Enter 512 or 4096,
          or a null response to use the original size,
          or 111(Quit) to exit.

512
ARI6103A Enter virtual address for new BDISK.
          (Enter a null response to end input or
          enter QUIT to exit.)

300
DMSACP112S C(300) device error
ARI6148I New disk 300 has not been formatted. Program will continue
          to format the disk before copying.
ARI6146D Are you expanding the DB2 Server for VM directory?
          Enter 0(No), 1(Yes), or 111(Quit).

1
ARI6118I Formatting in progress. Please wait...
ARI6130I Disk 300 is formatted successfully.
ARI6200D You have requested to expand the directory.
          Enter 1 to expand the directory to hold more DBSPACE
          pages. Enter 2 to expand the directory to hold more
          DBSPACE and DBEXTENT pages. Enter 111 to quit.
          Enter 1,2 or 111(Quit).

2
ARI6198D Current maximum DBEXTENT pages: 1662976
          New maximum DBEXTENT pages: 2502656
          DBEXTENT pages added: 839680
          Current maximum DBSPACE pages: 3294592
          New maximum DBSPACE pages: 4964352
          DBSPACE pages added: 1669760
          Current directory size: 27362
          Current directory block size: 512
          New directory size: 41046
          New directory block size: 512
          Do you wish to continue with expanding the
          directory to allow the directory to hold
          additional DBEXTENT and DBSPACE pages ?
          Enter 0(No), 1(Yes) or 111(Quit)

1

```

Figure 54 (Part 1 of 2). SQLCDBEX EXEC

```

ARI6131I Copying in progress. Please wait...
ARI0640I 4000 of
          41046 records copied to output disk.
ARI0640I 8000 of
          41046 records copied to output disk.
ARI0640I 12000 of
          41046 records copied to output disk.
ARI0640I 16000 of
          41046 records copied to output disk.
ARI0640I 20000 of
          41046 records copied to output disk.
ARI0640I 24000 of
          41046 records copied to output disk.
ARI0640I 28000 of
          41046 records copied to output disk.
ARI0640I 32000 of
          41046 records copied to output disk.
ARI0640I 36000 of
          41046 records copied to output disk.
ARI0640I 40000 of
          41046 records copied to output disk.
ARI6201I Directory expansion completed successfully.
          It is strongly recommended that a database
          archive be taken immediately.
ARI6108I Minidisk copied successfully. The SQLVM350 SQLFDEF file
          will be updated.
ARI6109I SQLVM350 SQLFDEF file has been updated on the A disk.

ARI6102A Enter DBEXTENT number (or LOGDSK1, LOGDSK2,
          or BDISK) to copy.
          (Enter a null response to end input or
          enter QUIT to exit.)

ARI0620I SQLVM350 SQLFDEF file
          successfully copied to production disk.
ARI0673I All COPY DBEXTENT processing completed successfully.
ARI0796I End SQLCDBEX EXEC: 09/28/95 17:11:35 CET
ARI0721I Get DB2 Server for VM production minidisk READ access: SQLDB350 195.
          Ready; T=90.80/106.77 17:11:40

```

Figure 54 (Part 2 of 2). SQLCDBEX EXEC

Acquiring Dbspaces for Packages

The process of adding a dspace merely reserves pages for it in the directory. Before it can actually be used, it must be *acquired*. For details of how to acquire dbspaces, see the *DB2 Server for VM Database Administration* manual.

Packages and view definitions are stored in system dbspaces named SYS0002, SYS0003, SYSnnnn. Allocation of the initial system dspace (SYS0002) is performed during database generation. You should probably acquire an additional package dspace after installation, and then more as needs arise. Because unused dbspaces only require minimal directory space and no data pages, acquiring them is not costly. Thus, if your installation has many packages and views, it is a good idea to acquire several dbspaces for packages in advance for later use.

The database manager stores packages and view definitions as tables. A dbspace can contain up to 255 tables, and can therefore have up to 255 packages and view definitions.

Although packages and view definitions are stored as tables, information about them is found not in the SYSTEM.SYSCATALOG catalog table, but in the SYSTEM.SYSACCESS catalog table. When a dbspace is acquired for packages, 255 empty tables are preallocated in it. For each table that is created, a row is added to the SYSTEM.SYSACCESS catalog table that identifies the package table as unused. Unused package tables can be either available or unavailable. The TNAME value in SYSACCESS for unused package tables is represented either as !0x AVAILABLE or ¢0x UNAVAILABLE. (The x is a number from 1 to 5, which is used internally.) Initially, all package tables in a newly acquired dbspace are unused and available. As packages are created and views are defined, the TNAME value is changed to indicate the package or view name.

As mentioned above, you can usually fit 255 packages in a dbspace. However, if large packages are created, the dbspace pages may fill before all 255 package tables are used. In this situation, all remaining package tables are unused and unavailable and their TNAME value is marked in the dbspace as ¢0x UNAVAILABLE. When the dbspace is full, the FREEPCT column of the SYSTEM.SYSDBSPACES catalog table is updated. A FREEPCT of 1 means that space is still available, while a FREEPCT of 0 means that this dbspace is full.

If a package or view is dropped from a dbspace that is not full, the database manager does not drop the package table from the dbspace. Instead, it deletes all the rows from the table, and marks the table as available in the SYSTEM.SYSACCESS catalog table. The table can then be reused.

If a package or view is dropped from a dbspace that has been marked as full (FREEPCT = 0), FREEPCT is reset to 1. Before these package tables can be reused, however, their TNAMEs in the SYSTEM.SYSACCESS catalog table must be changed to indicate that they are available. This is not done immediately, because if it were, the next time someone tried to create a package, the database manager would reuse the table from the package or view that was just dropped. It would try to place the newly created package in a dbspace that is almost full, and it probably would not fit. Thus, if you have used all the space in your package dbspaces, you should acquire another dbspace rather than try to free space by dropping one or two unused packages. The package tables will be marked available the next time the database manager does preallocation.

Preallocation is done when you acquire a new package dbspace. It is also done when you try to create a view or a new package, and there are no available packages. If the database manager cannot find an available package, it looks in all dbspaces that are not full (FREEPCT=1) for package tables that are marked unavailable, and marks them as available.

A user with DBA authority can acquire a package dbspace by issuing the following SQL statement when the database is running in multiple user mode:

```
ACQUIRE PUBLIC DBSPACE NAMED SYSnnnn (PAGES=xxxx)
```

where

nnnn is the number of the package dbspace. (SYS0002 is the initial dbspace, so the next one will be called SYS0003, the next one, SYS0004, and so on.).

xxxx is the number of pages of address space for the dbspace. The usual value is 2048, but you can set it larger or smaller if your programs have a large or small number of SQL statements in them, or if you are adding many views to the database.

You should specify the PAGES parameter because the default value of 128 is usually too small. You can specify NHEADER or allow it to default to 8. The database manager sets PCTFREE to 1, PCTINDEX to 0, and LOCK to PAGE (page locking). If you try to specify any of these parameters, your settings will be ignored.

If no package tables are available in any package dbspace during preprocessing, SQLCODE -945 is returned, and the DBA must acquire another dbspace for packages.

If sufficient space is not available in the dbspace where the database manager attempts to create the package, it returns SQLCODE -946. The user's response depends on the availability of package tables in other dbspaces. If some are available, the user can try to preprocess the program again. (The database manager does not choose the same dbspace again because it sets FREEPCT=0 when the preprocess fails.) If no package tables are available, another dbspace for packages must be acquired.

To get information about unused packages (available and unavailable), issue the following query:

```
SELECT * FROM SYSTEM.SYSACCESS WHERE TNAME LIKE '%AVAILABLE'
```

To determine which package dbspaces are full because all the space is taken, issue:

```
SELECT * FROM SYSTEM.SYSDBSPACES WHERE DBSPACENAME LIKE 'SYS0%'
```

If the FREEPCT value is 0, there is no free space in the dbspace.

To determine which package dbspaces are full because all 255 tables are occupied, issue:

```
SELECT DBSPACENO, COUNT(*) -  
FROM SYSTEM.SYSACCESS -  
WHERE TNAME NOT LIKE '%AVAILABLE' -  
GROUP BY DBSPACENO
```

Dbspaces with a count of 255 have no available package tables. (For information on the syntax of the ACQUIRE DBSPACE and SELECT statements, see the *DB2 Server for VSE & VM SQL Reference* manual.)

Managing Storage Pools

Typically, you set up your database to be supported by multiple storage pools, so that you can control what data resides on what devices, and can manage physical DASD allocations differently for different data. The following sections discuss uses of storage pools and how to define them.

Design Considerations for Storage Pools

A storage pool consists of a large collection of 4-kilobyte DASD pages, called *slots*, for storing allocated public and private dbspace pages and shadow pages (old copies of dbspace pages that have changed since the last checkpoint). Dbspace pages that are not allocated are not stored. For internal dbspaces, slots are occupied only by nonempty pages of data for active logical units of work.

The placement of dbspace pages in storage pool slots is determined by the database manager; however, you control which pool of slots the dbspace pages are assigned to. This allows you to control device utilization and the use of different DASD allocation schemes for different data.

Estimating Storage Requirements

You may often choose to *undercommit* the actual DASD space available for the dbspaces. Because a dbspace cannot be extended after it is defined, and because it is really only a logical allocation of space, many dbspaces are defined to be much larger than needed. As a result, the actual storage pool slots required are fewer than the dbspace sizes imply. The number of dbextent pages should be defined to support the expected number of dbspace pages that will actually be used.

The undercommitting approach to managing storage pools is particularly useful if the tables involved are expected to grow over time. The sizes of the dbspaces are set based on how large the tables can grow, while the size of the storage pool is defined based on current storage requirements. As the tables grow, you can extend the storage pool by adding dbextents to it.

Undercommitting is also useful for supporting internal dbspaces. It is unlikely that you will ever need all the pages of all of the internal dbspaces at the same time. The number of internal dbspaces defined is based on the most the database manager would need at one time, and the size for each is defined based on the worst possible situation that could occur. (Note that internal dbspaces are all the same size.)

If you want to guarantee space availability, or have more dynamic dbspace storage requirements, you should *overcommit* the DASD space available for dbspaces. For example, you might want to do so to handle the storage requirements for private dbspaces. User requests for more or bigger dbspaces can be relatively frequent. Rather than repeatedly going through an ADD DBEXTENT operation, you could overcommit the storage pool for private dbspaces and handle the user requests through the ADD DBSPACE and ACQUIRE DBSPACE operations. (You may still have to run the ADD DBEXTENT operation, but not as often.) For overcommitting, allocate sufficient slots to handle all dbspace pages plus the potential shadow pages.

Controlling Device and Channel Utilization

Storage pools enable you to control device and channel utilization through one of two basic approaches:

- Separating highly referenced dbspaces
Two highly active dbspaces can be placed on different devices by assigning them to different storage pools and defining the dbextents of these storage pools on different devices.
- Spreading a highly referenced dbspace across devices
A single highly active dbspace can be spread across multiple devices by defining its storage pool as small, multiple dbextents, each of which is a CMS minidisk defined on a different device.

Controlling Data Location

You can allocate a table and all its indexes to a specific device or CMS minidisk. To do this, create the table in a dbspace with no other tables, assign the dbspace to its own storage pool, and define the dbextent (or dbextents) as the CMS minidisk (or minidisks) on the volume that you want.

Monitoring Storage Pools

Use the SHOW POOL command to display physical storage information about each storage pool defined, including:

- The total number of pages in the storage pool
- The number of pages being used
- The percentage of the pages in use
- The number of dbextents defined for that storage pool, in the order in which they were defined (which is also the order in which they will be searched for a free page)
- For each dbextent
 - The total number of pages
 - The number of free pages
- A short-on-storage indicator.

You can issue the SHOW POOL command from either the operator console or from ISQL. For more information about it, refer to the *DB2 Server for VSE & VM Operation* manual. To see information about reusable deleted dbextent numbers, use the SHOW POOL DELETED command.

Maintaining Storage Pools

To maintain storage pools, you:

- Add storage pools to the database
You add a storage pool to a database by adding a dbextent to a nonexistent storage pool, using the ADD DBEXTENT process described in “Adding Dbextents to a Storage Pool” on page 166.
- Add storage to existing storage pool
If any of your storage pools are short on storage, you can use the ADD DBEXTENT process to increase their size.

- Remove storage from storage pools
You can use the DELETE DBEXTENT process to release DASD for other uses.
- Move dbextents to another device
- Move log disks to another device

Adding Dbextents to a Storage Pool

Dbextents can be added to a nonexistent storage pool (which defines a new storage pool), or to an existing storage pool (which increases the size of the storage pool) using the following two-step process:

1. Define a minidisk for each dbextent being added.
2. Run the SQLADBEX EXEC from the database machine.

These steps are described in more detail below.

Step 1: Define the Dbextent Minidisks: Before adding a dbextent to the database, you must define a minidisk for that dbextent. The minidisk definition allocates the DASD space and establishes the size of the dbextent. You define a minidisk by adding an MDISK control statement to the VM directory of the database machine.

Figure 55 shows example MDISK control statements for three minidisks that are to be database dbextents.

```
MDISK 31A 3380 cylr 50 DBDSK7 R DBX01 AFRT
MDISK 323 3380 cylr 20 DBDSK8 R DBX01 AFRT
MDISK 43A 3380 cylr 30 DBDSK8 R DBX01 AFRT
```

Figure 55. Example MDISK Statements for Adding Dbextents

Note: Refer to Table 43 on page 452 for minimum space allocation values.

In the example shown in Figure 55, one dbextent minidisk with a virtual device address of 31A is defined on volume DBDSK7. Two more, on virtual device addresses 323 and 43A, are defined on volume DBDSK8.

Read access mode (R) is specified with a read (DBX01) and write (AFRT) password for each minidisk. A user who knows the passwords can access the minidisks when the database manager is running in single user mode.

Database minidisks must always have a read password, a write password, and an access mode of R. If the passwords or access mode are overlooked, the minidisks are susceptible to careless or malicious access. For more information on the MDISK control statement, see the *VM/ESA: Planning and Administration* manual.

In the example, the sizes of the dbextent minidisks are specified in cylinders. The number of storage pool slots represented depends on the device types of DBDSK7 and DBDSK8. Because both are IBM 3380 volumes, the slots represented are:


```
virtual device 31A -- 7467 slots
virtual device 323 -- 2964 slots
virtual device 43A -- 4446 slots
```

1 slot = 1 4-kilobyte block

The tables in Appendix B, “Estimating Database Storage” on page 449 show how many slots are held on each of the different count-key-data (CKD) devices. For FB-512 devices, the sizes are specified in blocks. One dbextent page equals 8 blocks of an FB-512 device.

You can move dbextents between device types so long as the dbextent is not larger than the size of the device. When you define dbextents, you should keep this in mind. For example, if you defined a dbextent of 600000 blocks on a 9335, you could not move that dbextent to a 9332 (which is limited to 360032 blocks). If you defined 3 dbextents, each of 200000 blocks, on a 9335 (for a total of 600000 blocks), you could move them to three 9332 devices.

Updating the MAXCONN Setting: When adding MDISK control statements for a database machine, you must increase the MAXCONN value by the number of dbextents added. The MAXCONN value is a parameter of the VM OPTION control statement. This value determines the number of VM IUCV or APPC/VM connections allowed for a virtual machine. The MAXCONN parameter is unique to each database machine. For more information, see “Setting the MAXCONN Value” on page 298.

Step 2: Run the SQLADBEX EXEC: The SQLADBEX EXEC updates the database directory to include the control information for the dbextent. It also adds the appropriate CP LINK and CMS FILEDEF commands to the database SQLFDEF file. Multiple dbextents can be defined in one run of the SQLADBEX EXEC. For more information, see “Running the SQLADBEX EXEC” on page 171.

Example of Adding a Dbextent to a Database: Figure 56 on page 168 illustrates the sequence of commands required to add a dbextent to a database named TEST. It is added to storage pool 1 at disk address 307. The SQLADBEX EXEC then automatically calls the SQLSTART EXEC.

Before you run the SQLADBEX EXEC, you should know the disk addresses of the dbextents and the numbers of the pools to which the dbextents are being assigned. The SHOW DBEXTENT command indicates pool numbers and the number of dbextents currently defined.

The example assumes that the new minidisk has already been defined and added to the VM directory. The minidisk has been formatted and reserved. This step is optional, but it allows the minidisk to be formatted and reserved without stopping the application server. Entries you would make are indicated in the example.

```

1  → sqladbex db(test)
ARI0717I Start SQLADBEX EXEC: 01/20/93 10:49:52 EST.
ARI6111A Enter action (ADD or DELETE) to be taken.
      (Enter a null response to end input or
      enter QUIT to exit.)
2  → add
ARI6112A Enter DBEXTENT number to use for the new extent.
      The default is 3.
      (Enter a null response to use the default value or
      enter QUIT to exit.)
3  → 3
ARI0614A Enter virtual address and storage pool number
      (default = 1) of DBEXTENT 3.
4  → 307
ARI6110D Disk 307 is already formatted. Continuing will erase
      all data on this disk. Do you want to use the disk?
      Enter 0(No), 1(Yes), or 111(Quit).
5  → 1
ARI0647D Do you want to do a CMS FORMAT/RESERVE command on disk 307?
      Enter 0(No) or 1(Yes).
6  → 0
ARI6111A Enter action (ADD or DELETE) to be taken.
      (Enter a null response to end input or
      enter QUIT to exit.)
7  →
ARI6114A Do you want to do a database archive (ARCHIVE),
      user archive (UARCHIVE), or no archive (NOARCHIVE)
      at the end of the run?
      (Attention: Database may not be restorable
      if you choose NOARCHIVE.)
      Enter one of the values or enter a null response
      to use the default (ARCHIVE).
8  → archive
ARI6145D Do you want to review the SQLADBEX file?
      You will not be able to modify this file.
      Enter 0(No) or 1(Yes).
9  → 0

```

Figure 56. SQLADBEX Example of Adding a Dbextent

Notes for Figure 56:

- 1** Command to begin the ADD DBEXTENT operation. Because no parameters are specified, dcssid defaults to SQLDBA, POOL defaults to LOG, and PARM defaults to the values in Figure 11 on page 78.
- 2** add is entered to add dbextent
- 3** 3 is entered to add dbextent number 3.
- 4** The first dbextent added is located in storage pool 1 at disk address 307.
- 5** Disk 307 is correct so 1 (Yes) is entered.
- 6** Disk 307 is already formatted so 0 (No) is entered.
- 7** A null response is entered to end input.
- 8** archive is entered so an archive will be taken.

9 The file will not be reviewed, so 0 (No) is entered.

After **9**, the SQLSTART EXEC is automatically called. When this EXEC ends, the SQLADBSP EXEC also ends, and the dbextent has been added.

Deleting Dbextents from a Storage Pool

Deleting a dbextent does not delete any data in the database. Data in the deleted dbextent is moved to another dbextent in the same pool before the dbextent is removed from the database.

To delete dbextents:

1. Run the SQLADBEX EXEC from the database machine.

The SQLADBEX EXEC updates the database directory to remove the control information for the dbextent. It also deletes the appropriate CP LINK and CMS FILEDEF commands to the database SQLFDEF file. Multiple dbextents can be deleted in one run of the SQLADBEX EXEC.

For more information, see “Running the SQLADBEX EXEC” on page 171 and “Example of Deleting a Dbextent from a Database.”

2. Detach the minidisk for each dbextent being deleted.
3. After an archive has been taken, remove the minidisks of the dbextents being deleted from the VM directory of the database machine.

You can move a dbextent from one storage pool to another by deleting it and adding it back to the new pool; however, you cannot delete, add, and then delete the same dbextent in a single run.

Attention

You must not delete the only dbextent from the storage pool that contains the internal dbspaces.

Example of Deleting a Dbextent from a Database: Figure 57 on page 170 illustrates the sequence of commands required to delete a dbextent from a database named TEST. You can use the *SHOW POOL* command to determine pool numbers and the number of dbextents currently defined. The SQLADBEX EXEC then automatically calls the SQLSTART EXEC.

Entries you would make in the example are indicated by numbers in parentheses on the right.

```

1  → sqladbex db(test)
ARI0717I Start SQLADBEX EXEC: 01/20/93 10:46:36 EST.
ARI6111A Enter action (ADD or DELETE) to be taken.
      (Enter a null response to end input or
      enter QUIT to exit.)
2  → delete
ARI6113A Enter DBEXTENT number to delete.
      (Enter QUIT to exit.)
3  → 1
ARI6111A Enter action (ADD or DELETE) to be taken.
      (Enter a null response to end input or
      enter QUIT to exit.)
4  →
ARI6114A Do you want to do a database archive (ARCHIVE),
      user archive (UARCHIVE), or no archive (NOARCHIVE)
      at the end of the run?
      (Attention: Database may not be restorable
      if you choose NOARCHIVE.)
      Enter one of the values or enter a null response
      to use the default (ARCHIVE).
5  → archive
ARI6145D Do you want to review the SQLADBEX file?
      You will not be able to modify this file.
      Enter 0(No) or 1(Yes).
6  → 0

```

Figure 57. SQLADBEX Example of Deleting a Dbextent

Notes for Figure 57:

- 1** Command to begin the DELETE DBEXTENT operation. Because no parameters are specified, dcssid defaults to SQLDBA, POOL defaults to LOG, and PARM defaults to the values in Figure 11 on page 78.
- 2** delete is entered to delete dbextents.
- 3** Dbextent number 1 is to be deleted.
- 4** A null response ends the input.
- 5** Entering archive selects a database archive.
- 6** 0 (No) is entered to bypass the review.

After **6**, the SQLSTART EXEC is automatically called. When this EXEC ends, SQLADBEX also ends, and the dbextent has been deleted.

Considerations for the MAXCONN Setting

Deleted dbextents are sometimes counted in determining the MAXCONN setting. The MAXCONN value is a parameter of the VM OPTION control statement. This value determines the number of VM IUCV or APPC/VM connections allowed for a virtual machine. The MAXCONN parameter is unique to each database machine. For more information, see “Setting the MAXCONN Value” on page 298.

Running the SQLADBEX EXEC

The SQLADBEX EXEC starts the application server in single user mode with STARTUP=E. It also calls the ADD and DELETE DBEXTENT operations. The DELETE DBEXTENT operation removes control information in the database directory for the dbextents being deleted. The ADD DBEXTENT operation initializes control information in the database directory for the dbextents being added, and defines new storage pools as being recoverable or nonrecoverable.

The SQLADBEX EXEC resides on the service disk (V-disk) and can only be run from an application server. Figure 58 shows the format of the SQLADBEX EXEC.

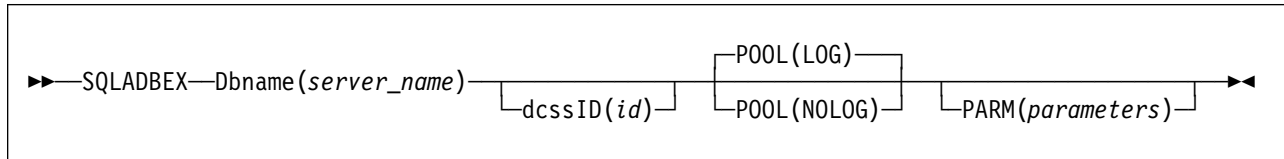


Figure 58. SQLADBEX EXEC

The parameters of SQLADBEX are as follows:

Dbname(server_name)

This parameter is required. You may use any initial substring as an abbreviation for the keyword. For *server_name*, specify the name of the database. (The name of the database is defined when the SQLDBINS EXEC is started to generate the database.)

dcSSID(id)

This parameter is optional. You can use DCSSID or ID for the keyword. For *id*, specify the name of the bootstrap package that identifies the saved segment. If not specified, the SQLDBA bootstrap package is used, and the database manager uses default saved segments. If you do not have default saved segments, DB2 Server for VM code is loaded into the user free storage area.

POOL(LOG) or POOL(NOLOG)

This parameter is optional. It is required only if you are defining a nonrecoverable storage pool with POOL(NOLOG). It is unnecessary if you are adding dbextents to existing pools because the status of the storage pools has already been defined.

If you specify POOL(NOLOG) to indicate that you want to define storage pools that are nonrecoverable, SQLADBEX prompts you for the numbers of any nonrecoverable storage pools you want to create. When prompted, you can respond with any value from 2 to the MAXPOOLS value for your database. The storage pools you select to be nonrecoverable must not already have dbextents assigned to them, and must not already have been defined in the SQLADBEX file.

If you omit the POOL parameter or specify POOL(LOG), which is the default, you are not prompted for the numbers of nonrecoverable storage pools. Nonrecoverable storage pools are described in “Nonrecoverable Storage Pools” on page 250.

PARM(parameters)

This parameter is optional. You use it to specify additional initialization parameters. Usually, the initialization parameters used by the SQLADBEX EXEC are sufficient. You can specify other initialization parameters as required.

If you specify the PARM parameter, it must follow the other SQLADBEX parameters. For a list of the valid initialization parameters, refer to Figure 11 on page 78. That figure lists the parameters that apply in single user mode. Do not specify the SYSMODE and STARTUP parameters. The SQLADBEX EXEC automatically supplies SYSMODE=S and STARTUP=E. Also, do not specify the PROGNAME, DUALLOG, and LOGMODE parameters. The SQLADBEX EXEC ignores any LOGMODE parameter. The LOGMODE setting is determined by other parameters specified on the SQLADBEX EXEC. See page 174 for more information on the LOGMODE setting.

You can specify the DUMPTYPE, TRACDBSS, TRACCONV, TRACDSC and TRACRDS parameters. For the definition of these parameters see “Multiple User Mode Initialization Parameters” on page 58. Because the ADD and DELETE DBEXTENT operations can be run only when the database manager is running in single user mode, the initialization parameters are the only means of tracing them. (Operator TRACE commands are only valid when the database manager operates in multiple user mode).

If you choose to use tracing, you may want to issue your own CMS FILEDEF and LABELDEF commands for the trace file. These optional FILEDEF and LABELDEF commands are discussed in the *DB2 Server for VM Diagnosis Guide and Reference* manual. More general information about tape capabilities is in “Tape Support” on page 79.

You can use PARMID to specify a CMS file that contains parameter specifications for the ADD or DELETE DBEXTENT operation.

For examples of using the SQLADBEX EXEC, see “Example of Adding a Dbextent to a Database” on page 167 and “Example of Deleting a Dbextent from a Database” on page 169.

The SQLADBEX processing has three parts:

1. Updating the *resid* SQLFDEF file to include CMS FILEDEF and CP LINK commands for the added dbextents. (Remember that SQLSTART uses *resid* SQLFDEF Q to access the database.) Server name and *resid* may be different. The RESID NAMES file on the production disk is used to map the *resid* to the server name defined during database generation.
2. Updating the database directory.
3. Updating the *resid* SQLFDEF file to remove the CMS FILEDEF and CP LINK commands for the deleted dbextents.

Updating the SQLFDEF File for Added Dbextents

When you start the SQLADBEX EXEC, it copies the file *resid* SQLFDEF from the production disk to the database machine A-disk. (Any file on the A-disk that has the name *resid* SQLFDEF is replaced.)

If you are doing an ADD DBEXTENT operation, the SQLADBEX EXEC prompts you for the dbextent number, the storage pool number and virtual device address

(*cuu*). If the minidisk has not been formatted and reserved, SQLADBEX will issue a CMS FORMAT and RESERVE command for it. If the minidisk is already formatted and reserved, SQLADBEX prompts you to proceed. Respond YES to the already formatted message displayed by SQLADBEX. The SQLADBEX EXEC prompts you to run the commands. You can choose to skip the FORMAT/RESERVE process if the minidisk has been previously formatted and reserved properly. Respond 1 (Yes) to run the commands or 0 (No) to skip them.

Attention

Be sure that you are accessing the correct minidisk before you respond 1 to the FORMAT and RESERVE notification.

The SQLADBEX EXEC then adds the appropriate CMS FILEDEF and CP LINK commands to the *resid* SQLFDEF file for the new dbextent.

When all the minidisks have been added SQLADBEX copies the updated *resid* SQLFDEF file to the production disk. The SQLFDEF file on the production disk is replaced.

If the action is delete, SQLADBEX prompts you for the dbextent number and optionally the storage pool number. The update to the *resid* SQLFDEF file is delayed until the update to the directory is done.

Updating the Database Directory

The SQLADBEX EXEC updates the database directory by using the ADD and DELETE DBEXTENT operations. The ADD and DELETE DBEXTENT operations require (as input) the specifications for the dbextents to be added and deleted. The EXEC generates the specifications for you, based on the storage pool numbers you provided in the previous step.

The SQLADBEX EXEC creates the file *resid* SQLADBEX on the database machine's A-disk. Figure 59 on page 174 shows the format of a *resid* SQLADBEX file. Any existing file with the name *resid* SQLADBEX is erased from the database machine's A-disk.

If you specify POOL(NOLOG) when running SQLADBEX, you are prompted for the numbers of storage pools that you want to define as nonrecoverable. Based on your responses to the prompts, SQLADBEX creates POOL control statements and inserts them in the *resid* SQLADBEX file. These POOL control statements are used by the ADD DBEXTENT operation to define nonrecoverable storage pools. Do not supply the numbers of storage pools that have already been defined. After definition, a storage pool cannot have its recovery status changed. Even if the storage pool contains no dbextents, once you have defined it as nonrecoverable, you cannot redefine it as recoverable. To see the storage pools that have been defined, use the SHOW POOL ALL command. Figure 59 on page 174 shows a POOL control statement that defines storage pool 8 as nonrecoverable. A subsequent control statement, which was also generated because of responses to other prompts, assigns dbextent number 6 to the storage pool.

As soon as the file is created, you are given the opportunity to review it.

```
POOL 8 NOLOG
DELETE 3 1
DELETE 2 2
ADD 6 8
DELETE 4
ARCHIVE
```

Figure 59. Format of the resid SQLADBEX File

The ARCHIVE control statement must be the last statement if present. The valid options are ARCHIVE (database archive), UARCHIVE (user archive) or NOARCHIVE (no archive). If the ARCHIVE control statement is not specified, the default (ARCHIVE) is used.

Attention

After a dbextent is deleted, the database cannot be restored from an archive taken prior to the deletion.

Therefore, the user should choose ARCHIVE or UARCHIVE to backup the database. If NOARCHIVE is chosen, the LOGMODE will be switched to Y. The LOGMODE parameter is set to A if ARCHIVE or UARCHIVE is chosen.

When the *resid* SQLADBEX file is complete, the SQLADBEX EXEC starts the ADD and DELETE DBEXTENT operation.

The optional POOL control statements must precede the statements that define the dbextents. They are required only for defining new nonrecoverable storage pools with POOL(NOLOG). They are unnecessary if you are adding dbextents to an existing pool because a storage pool's status has already been defined as either nonrecoverable or recoverable. POOL statements are also not necessary for new recoverable storage pools, because by default, storage pools are recoverable.

You cannot specify pool number 1 on any POOL control statement.

Attention

Once a storage pool is defined as either nonrecoverable or recoverable, you must not change it from recoverable to nonrecoverable (or from nonrecoverable to recoverable).

The records following the POOL control statements contain the dbextent definitions. Each control statement must contain a control word (ADD or DELETE) and the specification of one dbextent. The first number in the input record is the number designator of the dbextent. The second number, if specified, is the number designator of its storage pool. (For the ADD action, if this number is not specified, the default is storage pool 1; for the DELETE action, the default is the storage pool where the dbextent resides.) The numbers must be separated by at least one blank.

When you add a dbextent, its number must either be one more than the number of dbextents currently defined, or the number of any dbextent that was deleted by the DELETE DBEXTENT operation. The total amount of space allocated in the

directory as the dbextent control area is fixed for a database, and cannot be changed without regenerating the database. When a dbextent is deleted, the control area is not compressed. Therefore, you should reuse deleted dbextent numbers whenever possible so as to reuse the directory control area. Figure 60 shows the dbextent control area in the directory.

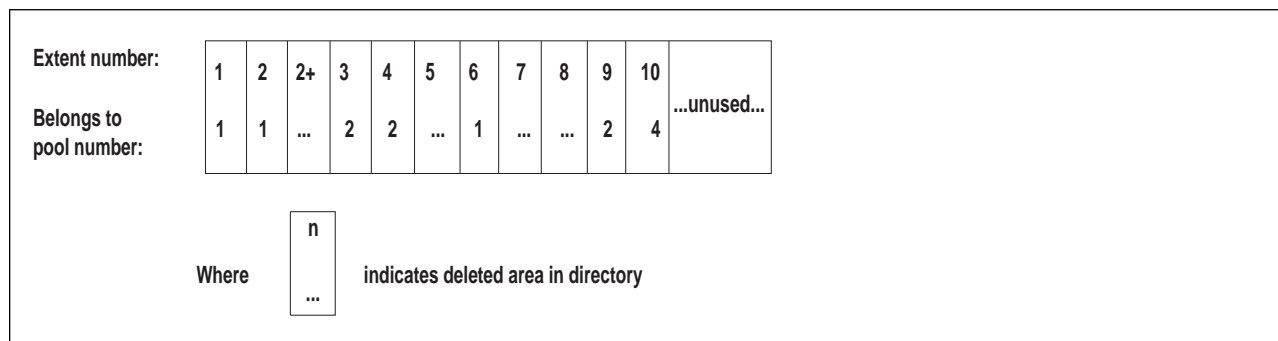


Figure 60. Dbextent Control Area in the Database Directory

In this example, a new dbextent can take on the numbers 5, 7 or 8, which are available for reuse, or 11, which is the next sequential number. The value 2+ indicates that there is empty directory space between dbextents 2 and 3. Because no dbextent number is associated with this space, you must first delete dbextent 2 or dbextent 3 to reclaim it.

You can determine the number of dbextents currently defined in a database by using the SHOW POOL operator command. To determine the maximum number of dbextents or storage pools that can be defined for the database, issue the SHOW DBCONFIG operator command. For more information, see the *DB2 Server for VSE & VM Operation* manual.

You can determine the deleted dbextent numbers that are available to be reused by using the SHOW POOL DELETED command. There is a maximum size associated with each deleted dbextent number. The maximum size is determined by the previous use of the dbextent number. The highest number is an exception; if it is deleted, the control area it used to occupy will be combined with the rest of the free area and this number will be treated as if it has never been used.

For example, if dbextent 10 in Figure 60 above is deleted, the control area in the directory will look like Figure 61.

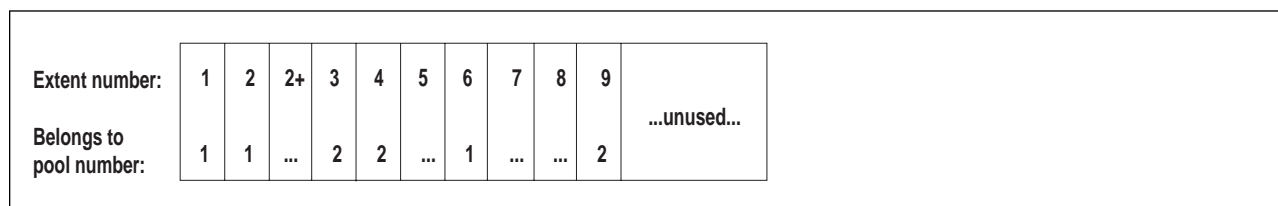


Figure 61. Dbextent Control Area in the Directory after Dbextent 10 Is Deleted

When the SHOW POOL DELETED command is issued, dbextent number 10 will not be listed.

The storage pool numbers you enter can range from 1 to MAXPOOLS, where MAXPOOLS is the maximum number of storage pools for the database as specified

during the database generation. You can use the storage pool numbers in any sequence.

For a dbextent to be deleted, the dbextent number must be one of the dbextents currently defined to the database. If the storage pool number is specified, it must be where the dbextent resides.

After the file *resid* SQLADBEX is created and reviewed, the SQLADBEX EXEC starts the application server in single user mode with the ADD or DELETE DBEXTENT operation. When the application server ends, the dbextents are deleted, or added and ready to be used.

Updating the SQLFDEF File for Deleted Dbextents

The SQLADBEX EXEC removes CMS FILEDEF and CP LINK commands for all deleted dbextents from the *resid* SQLFDEF file.

When all the minidisks have been deleted, SQLADBEX copies the updated *resid* SQLFDEF file to the production disk. The SQLFDEF file on the production disk is replaced, and SQLADBEX then erases the *resid* SQLFDEF file from your A-disk.

Possible Outcomes

Message ARI0620I *resid* SQLFDEF successfully copied to production disk indicates the successful completion of the ADD and DELETE DBEXTENT operation. If the operation does not complete successfully, the action you take depends on what part of the processing failed:

- If SQLACDBEX fails because of incorrect control statement input, rerun it after correcting the cause of the error
- If a failure occurs after message ARI0717I Start SQLSTART EXEC:, refer to Table 14 for the required action depending on which messages you have received.

Table 14. Recovering from Errors during SQLADBEX

Messages Issued				
ARI0922I	ARI0650E	ARI0620I	Action	Notes
No	Yes	No	None required; you can rerun SQLADBEX if you want.	Failed. SQLFDEF file not updated. Directory not updated.
Yes	Yes	No	You must rerun SQLADBEX with the same input.	Failed. SQLFDEF file not updated. Directory updated.
Yes	No	No	You must rerun SQLADBEX and reply YES to ARI0646D or delete the dbextents that you tried to delete.	Failed. SQLFDEF file updated for added dbextents only. Directory updated.
No	No	No	You must rerun SQLADBEX and reply YES to ARI0646D or delete the dbextents that you tried to add.	Failed. SQLFDEF file updated for added dbextents only. Directory not updated.

Considerations for Adding and Deleting Dbextents

Neither the ADD nor the DELETE DBEXTENT operation is recorded in the log. Because these operations update the directory, and not the database itself, you can encounter a problem if you normally archive the database, and then try to restore it. For an ADD DBEXTENT operation, suppose the following events occur in the following order:

1. You do a database archive
2. You add dbextents
3. Users use data from those dbextents
4. You do an archive restore using the archive file from number 1 above and, if you use LOGMODE=L, subsequent log archives.

The directory and the database are not synchronized. The directory was restored from an archive file that did not reflect the ADD DBEXTENT operation; the database is also restored from that file however, the use of the changed dbextents is also restored from updates recorded in the log or log archives. Thus, the directory does not reflect the changed dbextents, but the database does.

For a DELETE DBEXTENT operation, suppose the following occurs:

1. You do a database archive
2. Later you delete dbextents
3. You attempt to do an archive restore from number 1 above.

The restore operation fails because it attempts to put data on the dbextents that have been removed.

You can prevent this problem by using the ARCHIVE or UARCHIVE option in the ADD or DELETE DBEXTENT operation. This will ensure that your current database archive reflects the changed dbextents.

The same problems occur if you use log archiving and restore the database using a database archive taken before the ADD or DELETE DBEXTENT operation. That is, if you use a back-level database archive and subsequent log archives to restore the database, the database archive that records the changes to the dbextents are skipped. For ADD DBEXTENT operations, the directory, restored from the back-level database archive, does not show the changes to the dbextents; the subsequent log archives, however, do record the use of those dbextents. Restoring the database from an old database archive and subsequent log archives can thus put the database out of synchronization with the directory. For DELETE DBEXTENT operations, the restore fails when it tries to use the removed dbextents.

Moving Dbextents

Sometimes you must relocate the dbextents to another device because of disk migration or to control device utilization. A dbextent can be moved using the SQLCDBEX EXEC. The “move” is actually a “copy.” The dbextent is copied from one device to another. The dbextent remains on the old device until the copy is successfully committed, then the old device is released by the database manager. At this point, the dbextent has been moved.

The new device should be the same size as the old device. Moving a dbextent to a larger device does not expand the size of the dbextent. The extra space available

on the larger device is not available to the dbextent. If you need to increase the size of your database, you must use the SQLADBEX EXEC. For more information on this EXEC, see “Adding Dbextents to a Storage Pool” on page 166.

The SQLCDBEX EXEC invokes the application server in single user mode, so before a dbextent can be moved, the application server must be shut down. The EXEC is located on the service disk.

Figure 62 shows the format of the SQLCDBEX EXEC.

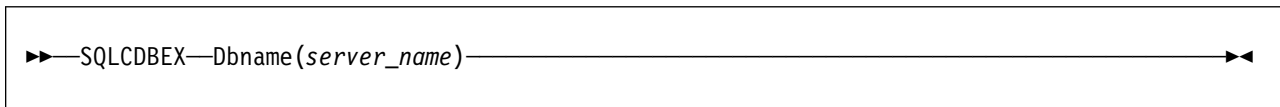


Figure 62. SQLCDBEX EXEC

Dbname(server_name)

The DBNAME parameter is required. Any initial substring for DBNAME can be used as the keyword (for example, DB or D). For *server_name*, specify the name of the application server. (The name of the application server is defined when the SQLDBINS EXEC is started to generate the database.)

When the EXEC is run, it prompts for all the information it requires to carry out the operation. The dbextents are copied to the new devices defined and the *resid* SQLFDEF file is updated on the A-disk.

The changes to the database are committed when all the copies have been performed successfully and the user indicates to end the EXEC. If the system crashes or the user quits from the EXEC, the dbextents remain on the old devices.

The old devices are released by the database manager when the changes are committed.

The SQLCDBEX EXEC can be used to move the directory.

Example of Moving a Dbextent

Figure 63 illustrates the sequence of commands required to move a dbextent. The example assumes that the new minidisk has already been defined and added to the VM directory. The step to format and reserve the minidisk does not need to be performed if the minidisk has been formatted and reserved before running the EXEC.

```

1  -> sqlcdbex db(sqldba)
ARI0717I Start SQLCDBEX EXEC: 01/19/93 18:01:47 EST.
ARI0721I Get DB2 Server for VM production minidisk WRITE access: SQLDBA 195.
ARI6102A Enter DBEXTENT number (or LOGDSK1, LOGDSK2,
         or BDISK) to copy.
         (Enter a null response to end input or
         enter QUIT to exit.)

2  -> 2
ARI6103A Enter virtual address for new DBEXTENT 2.
         (Enter a null response to end input or
         enter QUIT to exit.)

3  -> 204
ARI6110D Disk 204 is already formatted. Continuing will erase
         all data on this disk. Do you want to use the disk?
         Enter 0(No), 1(Yes), or 111(Quit).

4  -> 1
ARI0647D Do you want to do a CMS FORMAT/RESERVE command on disk 204?
         Enter 0(No) or 1(Yes).

5  -> 0
ARI6131I Copying in progress. Please wait...
ARI6108I Minidisk copied successfully. The SQLDBA SQLFDEF file
         will be updated.
ARI6109I SQLDBA SQLFDEF file has been updated on the A disk.
ARI6102A Enter DBEXTENT number (or LOGDSK1, LOGDSK2,
         or BDISK) to copy.
         (Enter a null response to end input or
         enter QUIT to exit.)

6  ->
ARI0620I SQLDBA SQLFDEF file
         successfully copied to production disk.
ARI0673I All COPY DBEXTENT processing completed successfully.
ARI0796I End SQLCDBEX EXEC: 01/19/93 18:03:12 EST
ARI0721I Get DB2 Server for VM production minidisk READ access: SQLDBA 195.

```

Figure 63. SQLCDBEX Example of Moving a Dbextent

Notes for Figure 63:

- 1** Command to start the MOVE DBEXTENT operation.
- 2** Dbextent 2 is to be moved.
- 3** Disk 204 is the new device address.
- 4** Disk 204 is correct so 1 (Yes) is entered.
- 5** Disk 204 has been formatted and reserved using CMS FORMAT and RESERVE prior to the invocation of the EXEC, so 0 (No) is entered. Note that if you have access to the DFSMS/VM* product, message ARI0647D will not be displayed.
- 6** A null entry ends input.

Moving Log Disks

Sometimes you must relocate the log disks to another device because of disk migration or to control device utilization. The SQLCDBEX EXEC can be used to copy the log disk only if:

- 1. The target log disk is the identical device type and size as the source log disk
- 2. The source log disk is not damaged.

If these conditions are met, SQLCDBEX can be used to make an exact copy of the original log disk, and it is not necessary to reformat or reconfigure the log. If these conditions are not met, you must do a COLDLOG to reconfigure the new log disk.

For more information about COLDLOG, see “Reconfiguring and Reformatting the Logs” on page 240.

Chapter 8. Saved Segments

Saved segments allow users to share code. This sharing can reduce the amount of user free storage required by the user and the database machines, and reduces the amount of paging done by the system.

This chapter discusses installing these common areas.

Using Saved Segments for Components

You can define database manager components in saved segments. This can be done after you have installed the product. Saved segments allow code to be shared among users. Code that is not in saved segments runs in the user free storage area of the user machine and database machine.

Code can be loaded into the following saved segments:

Resource adapter

The resource adapter saved segment may contain the code for the following components:

- Resource Adapter (RA)
- DRRM
- CONV

The DRRM and CONV components are only applicable if the DRDA code is installed.

The resource adapter segment can be saved above 16M.

DBSS

The DBSS saved segment contains the code for the following components:

- DBSS
- DSC

The DBSS segment must be saved below 16M.

RDS

The RDS saved segment may contain the code for the following components:

- RDS
- WUM
- DRRM
- CONV

The DRRM and WUM components are only applicable if the DRDA code is installed.

The RDS Segment can be saved above 16M. This includes the WUM, DRRM and CONV components. If RDS is saved above 16M, the "AMODE(24)" initialization parameter **CANNOT** be used. If "AMODE(24)" is specified with the RDS Segment defined above 16M (or if RDS is loaded into free storage above 16M), message ARI0021I is issued and start up fails. If you must use the "AMODE(24)" parameter, you must create an alternative bootstrap package which specifies an RDS segment that is saved below 16M. If RDS is not used

in a saved segment, and "AMODE(24)" is required, you must define the virtual machine storage to 16M or less, to prevent RDS from being loaded above 16M.

ISQL

The ISQL saved segment contains the ISQL code. It must be saved below 16M.

National language message repositories

This saved segment contains the code for the message repository. It must be saved below 16M.

Notes:

1. The DBSS and RDS components must both reside in saved segments or both reside in the user free storage area.
2. Any segments that are within the database machine's virtual storage should be reserved using the SEGMENT RESERVE command. For more information on the SEGMENT RESERVE command, see the *VM/ESA: CMS Command Reference*.

Use the VMFSGMAP EXEC, which uses the VM/ESA CP DEFSEG command, to define each component in a saved segment. After defining the saved segments you must load them (using the VMFBLD EXEC, which uses the ARISAVES EXEC), and create a bootstrap package to use them (using the SQLGENLD EXEC). If you choose to have ARISAVES create the default (SQLDBA) bootstrap package, it is not necessary to run SQLGENLD.

In any bootstrap package there can be three bootstrap modules that correspond to the following components:

- Resource adapter
- ISQL
- DBSS and RDS.

The bootstraps identify where the components reside. Only one bootstrap module is needed for the DBSS and RDS components. Because the DBSS and RDS components must both reside in saved segments or both reside in user free storage, only one module is needed to indicate the location of the code. (The DBSS and RDS code is often referred to as the DB2 Server for VM system code.) For a service machine, only the bootstrap modules for the resource adapter and ISQL need to be generated.

Each set of saved segments defined for the database manager usually has three bootstrap modules. (Typically, only one set of saved segments is defined for the code at an installation.) The bootstrap modules identify the corresponding saved segments. The database manager requires bootstrap modules for every saved segment except national language message repositories. National language message repository saved segments are identified in the ARISNLSC MACRO.

The bootstrap modules allow the database manager to use the code residing in the saved segments. The SQLBOOTS EXEC generates the appropriate bootstrap modules (the SQLDBA bootstrap package) for default saved segments. If you respond YES when the ARISAVES EXEC prompts you to use the saved segments you are loading as defaults, ARISAVES calls SQLBOOTS to create the SQLDBA bootstrap package. This bootstrap package specifies whether the database

manager code runs as a default in saved segments or not. If not, the code runs in the user free storage area. ISQL and the resource adapter run in the user free storage area of the user machines, and the system code runs in the user free storage area of the database machines. Also, the resource adapter runs in the user free area of the database machine when in single user mode. To create other bootstrap packages, you use the SQLGENLD EXEC. You name the bootstrap package when you use SQLGENLD to create it. For more information on the SQLGENLD EXEC and the SQLBOOTS EXEC, see “Defining Saved Segments” on page 187.

Even though you have defined one or more saved segments for the database manager, they are used only in the following situations:

- They are default saved segments, and the database manager has generated the SQLDBA bootstrap modules for them.
- You have created bootstrap modules for them, and specify that bootstrap package when you run DB2 Server for VM EXECs.

You indicate you want to use a particular saved segment by specifying the DCSSID parameter on various IBM-supplied EXECs (for example, the SQLSTART EXEC and the SQLINIT EXEC). Indicate that you want to use the saved segments by specifying the name of the corresponding bootstrap package. (Remember, you name a bootstrap package when you create it using SQLGENLD.) After you use the DCSSID parameter, the EXEC continues to use that bootstrap package until you specify another bootstrap package. If you do not specify anything on the DCSSID parameter, and have never used it, you use the SQLDBA bootstrap package. The SQLDBA bootstrap package identifies default saved segments, if you have them. If not, the SQLDBA bootstrap package specifies that the database manager code runs in the user free storage area.

If you omit the DCSSID parameter after defining additional bootstrap packages, the default rules are more complex. They are even more complex when you do not define all bootstrap modules in a bootstrap package. (Consider doing this if, for example, you want the system code to run in special saved segments, but want ISQL and the resource adapter to run in default saved segments.) The following sections describe the default rules.

If you want to keep things simple, define only one set of saved segments and generate three bootstrap modules. This is what is done for default saved segments. If you choose to have default saved segments, you define them, and the database manager creates a bootstrap package for them. If you have default saved segments, they are used when your users run EXECs without specifying the DCSSID parameter. This way, you always easily know where the database manager code is running. For an example of defining saved segments and generating bootstraps, see “Defining Saved Segments” on page 187.

Even if you have default saved segments, you should still define saved segments for national language messages that you expect to use. A default saved segment is not generated for national language messages. For more information, see “Defining Message Repositories as Saved Segments” on page 351.

You may want to define multiple saved segments for the database manager code. (For example, you want to run the code at different locations.) The above guideline still applies: if possible, generate all three bootstraps, and provide the users with specific instructions for DCSSID parameters.

For a service machine, you only need to generate the bootstrap modules for RA and ISQL. The DBSS and RDS code cannot run from a service machine.

If you need more details on the way bootstraps work, continue reading. If not, skip the rest of this section.

The bootstrap module identifies the name of the saved segment to be loaded into storage at run time. In addition, the bootstrap module for the resource adapter also contains the name of the database machine. This name is required to establish the communication link in multiple user mode. In a VM/ESA operating system, you do not need to specify the name of the database machine because APPC/VM is used for communication: only the RESID (resource) is needed. After installation, there is one bootstrap package made up of the following CMS files:

```
SQLDBA SQLDBBT Q -- corresponds to the DB2 Server for VM system code
                  (not generated for a service machine)
SQLDBA SQLISBT Q -- corresponds to the ISQL code
SQLDBA SQLRMBT Q -- corresponds to the resource adapter code
```

The files are created at installation time and reside on the production minidisk. Collectively, they form the SQLDBA bootstrap package.

When you run the SQLGENLD EXEC to create a bootstrap package, you are prompted for the name to be specified in the DCSSID parameter and the name (or names) of the saved segment (or saved segments). The name to be specified in the DCSSID parameter is used to identify the bootstrap package. For example, if you generate a bootstrap package with the name MYBOOT for the system code, ISQL, and the resource adapter, these files are created on the production minidisk:

```
MYBOOT SQLDBBT -- DB2 Server for VM system code bootstrap module
MYBOOT SQLISBT -- ISQL bootstrap module
MYBOOT SQLRMBT -- resource adapter bootstrap module
```

When you run the SQLSTART EXEC, the bootstrap for the system code and resource adapter are copied to the A-disk of the database machine. (Because you can run the database manager in single user mode, and it cannot determine the mode you will use, the resource adapter bootstrap is always copied.) Assume that you issue the following command, and DBNAME and RESID are the same:

```
SQLSTART DBNAME(TEST1) DCSSID(MYBOOT) ...
```

The file TEST1 SQLDBN is either created or updated at this time, with the following information:

```
DBMACHID = name of the database machine
DCSSID   = MYBOOT
DBNAME   = TEST1
```

When you run the SQLINIT EXEC for the user machine, the bootstraps for ISQL and the resource adapter are copied to the A-disk of the user machine. Assume that you issue the following command:

```
SQLINIT DBNAME(TEST1) DCSSID(MYBOOT)
```

During processing, the resource adapter bootstrap is recreated with the name of the database machine. The TEST1 SQLDBN file is read to obtain the name of the database machine. This means you must have used the SQLSTART EXEC (or the SQLDBINS EXEC) to create the SQLDBN file for the database machine.

The *resid* SQLDBN file, containing the default DCSSID, cannot always be accessed; therefore, the default DCSSID in this file cannot be used. A user machine cannot access the *resid* SQLDBN file on a database machine in either of the following situations:

- The database machine resides on a different processor
- The database machine does not own the production minidisk (Q-disk) to which the user machine has a link.

You must create a new file (SQLDCSID DEFAULT) on the production (Q) disk that is linked by these user machines to provide a default DCSSID for them. This file is created when the SQLGENLD EXEC is run. The SQLGENLD EXEC generates the bootstrap package for a particular saved segment. The EXEC prompts you to specify whether the DCSSID is to be the default for user machines that have a link to this Q-disk.

The following examples outline the above discussion. The examples assume that you have default saved segments identified in the SQLDBA bootstrap package.

Example 1

Assume that you have created four saved segments (one for each of the following): the DBSS code, the RDS code, the ISQL code, and for the resource adapter code. (Also assume that you used the names SQLSQLDS for the DBSS/DSC, SQLXRDS for the RDS, SQLISQL for ISQL, and SQLRMGR for the resource adapter as the names of the saved segments.) You then used SQLGENLD to create the three bootstrap modules and identified this bootstrap package with the name MYBOOT. Assume that the name of the database machine is SQLMACH1 and that both DBNAME and RESID are TEST1. Until you specify the DCSSID parameter on the SQLSTART EXEC (or unless you provided it when you generated the database using the SQLDBINS EXEC), the code is loaded as specified by the SQLDBA bootstrap modules. The SQLDBN file has the following information:

```
DBMACHID = SQLMACH1
DCSSID   = SQLDBA
DBNAME   = TEST1
```

If a user machine that is linked to the database machine Q-disk containing the TEST1 SQLDBN file runs the SQLINIT EXEC, that EXEC is called to use the SQLDBA bootstrap modules. Assume that you run the SQLINIT EXEC as follows:

```
SQLINIT DBNAME(TEST1) DCSSID(MYBOOT)
```

During processing, the bootstraps for ISQL and the resource adapter use the saved segments SQLISQL and SQLRMGR. These saved segments are used even though the database manager is using the default bootstrap modules.

When the SQLSTART EXEC (or SQLDBINS EXEC) has been run with the DCSSID parameter DCSSID(MYBOOT), the database manager uses the saved segments SQLSQLDS and SQLXRDS. The user only has to specify the DBNAME parameter on the SQLINIT EXEC, for example, SQLINIT DBNAME(TEST1), to use the SQLISQL and SQLRMGR code.

Example 2

Assume you have created a second resource adapter saved segment named SQLRMGR2 using the bootstrap name RMBOOT2, and the environment has already been established to use MYBOOT as shown in “Example 1” on page 185. Also assume that DBNAME and RESID are the same, and that you start the application server as follows:

```
SQLSTART DBNAME(TEST1)
```

Notice that the DCSSID is not specified. The SQLSTART EXEC reads the TEST1 SQLDBN file and uses the MYBOOT bootstrap module to load the SQLSQLDS and SQLXRDS code into saved segments. In this situation, MYBOOT (rather than SQLDBA) has become the established default bootstrap package for *server-name* TEST1. When you run the SQLINIT EXEC specifying DCSSID (RMBOOT2) in a user machine that has a link to the Q-disk containing the TEST1 SQLDBN file, the bootstrap package exists for the resource adapter, but not for ISQL. For this example, run the SQLINIT EXEC as follows:

```
SQLINIT DBNAME(TEST1) DCSSID(RMBOOT2)
```

During processing, the SQLINIT EXEC regenerates a resource adapter bootstrap to load the saved segment named SQLRMGR2. Because it does not find a bootstrap identified by RMBOOT2 for ISQL, it reads the TEST1 SQLDBN file, finds the DCSSID=MYBOOT entry, and uses the bootstrap identified by MYBOOT for ISQL.

Example 3

Assume that you have a second database (TEST2) that is owned by the SQLMACH2 database machine, and that the database has a bootstrap package (SQLBOOT2) for the DB2 Server for VM system code only. Also assume that DBNAME and RESID are the same.

The TEST2 SQLDBN file has the following entries:

```
DBMACHID = SQLMACH2
DCSSID   = SQLBOOT2
DBNAME   = TEST2
```

Suppose that you run the SQLINIT EXEC using the DCSSID of RMBOOT2, as shown in “Example 2”:

```
SQLINIT DBNAME(TEST2) DCSSID(RMBOOT2)
```

During processing the bootstrap package for the resource adapter exists and is created on the user’s A-disk to communicate with the SQLMACH2 machine and *server-name* TEST2. If the SQLINIT EXEC does not find a bootstrap identified by RMBOOT2 for ISQL, or one identified by SQLBOOT2 (from TEST2 SQLDBN), or if the ARISISBT module is not found, it defaults to the SQLDBA ISQL bootstrap module.

Example 4

Note: This example only applies to systems with multiple databases in which default saved segments are not used.

Suppose you have users on a processor that does not have a database machine, and a service machine is defined as the owner of the Q-disk. You want the

resource adapter and ISQL code to run in saved segments. When creating the bootstrap package (BOOTS), using the SQLGENLD EXEC, answer YES to the following prompt:

Do you want B00TS to be the default DCSSID for user machines?

When you answer YES, a new file, SQLDCSID DEFAULT, is created on the production (Q) disk with the following entry:

DCSSID=B00TS

If DBNAME and RESID are the same, and the users run the SQLINIT EXEC without specifying the DCSSID parameter, the DCSSID in the SQLDCSID DEFAULT file is used as the default. Assume that a user enters the following command:

SQLINIT DBNAME (TEST2)

During processing, the bootstrap package BOOTS (for the resource adapter and ISQL) is copied to the user's A-disk.

When a user runs the SQLINIT EXEC without specifying DCSSID (BOOTS), and you have not specified that BOOTS is to be the default DCSSID for user machines, the SQLDBA bootstrap package is used.

You should specify that the bootstrap package for the resource adapter and ISQL (BOOTS in this example) is the default DCSSID when you create the bootstrap package (BOOTS) with the SQLGENLD EXEC. Doing this ensures that user machines not having a link to the Q-disk that contains the SQLDBN file for the database machine that they are accessing will have a default DCSSID. These user machines then do not have to specify the DCSSID parameter.

Defining Saved Segments

Table 15 shows the saved segment usage by virtual machine.

Saved Segment	User Machine	Database Manager in Multiple User Mode	Database Manager in Single User Mode
ISQL	X		
DBSS		X	X
RDS		X	X
RA	X		X
DB2 Server for VM message repository	X		
CMS DB2 Server for VM message repository	X	X	X

It is possible to overlap saved segments that will not be used in the same machine. For example, in multiple user mode, the DBSS and RDS components are used only by the database machine, and ISQL and the resource adapter are used only by the

user machine. Therefore, in multiple user mode, it is possible to overlay DBSS (or RDS) with ISQL (or the resource adapter).

Note: In single user mode, the resource adapter is used by the database machine, and must run in the same machine as DBSS and RDS, and therefore cannot overlap either RDS or DBSS.

Table 16 shows the components that can overlap when they are being run in multiple user mode *only*.

<i>Table 16. Shared Segment Relationships in Multiple User Mode</i>			
Component Name	Used By	Default SYSNAME	Segments This Component Can Overlap
Resource adapter	End users	SQLRMGR	SQLXRDS SQLSQLDS
ISQL	End users	SQLISQL	SQLXRDS SQLSQLDS
DBSS	Database manager	SQLSQLDS	SQLRMGR SQLISQL LANGS001
RDS	Database manager	SQLXRDS	SQLRMGR SQLISQL LANGS001
DB2 Server for VM Message Repository	End users Database manager	LANGS001	SQLXRDS SQLSQLDS

This section outlines the steps you must follow to put database manager code into saved segments. It is only a supplement to the information about defining saved segments in the manuals below. It assumes that you are familiar with saved segments and the procedures that must be followed to define them. For more information about saved segments, refer to the the *VM/ESA: Planning and Administration* manual.

To use saved segments for database manager component load modules, you must do the following:

1. Log on to the installation user ID, 5648A70S.
2. Run the VMFSGMAP command to add or change DB2 Server for VM segment definitions.
3. Ensure that the machine's virtual storage is defined large enough to contain the segments to be loaded. It must have sufficient storage to contain the saved segment, loader tables, and CMS control block storage at the end of virtual storage.
4. Ensure that you have write access to the database machine production disk or SFS directory.
5. Issue the SET LANGUAGE command to ensure that the language repository is available.
6. Verify and update the ARISSEGC macro file.

7. Run VMFBLD EXEC, which calls ARISAVES EXEC, to load and save each segment.
8. Run the IBM-supplied SQLGENLD EXEC to generate a bootstrap package for the saved segment.

This step does not need to be performed if you are creating or modifying default saved segments (that is the saved segments used by the SQLDBA bootstrap package). When you run ARISAVES for default saved segments, the database manager automatically creates a bootstrap package (SQLDBA) for you. If you are defining other saved segments, after you run ARISAVES, you must run SQLGENLD.

9. Reset the virtual storage to its original value or issue the SEGMENT RESERVE command for any segments that are within the database machine's virtual storage.

The following is an example of the process used to define saved segments for the components. With the VM/ESA operating system, you must define the segments with VMSES/E VMFSGMAP EXEC, which calls the DEFSEG command.

Step 1. Plan to Define and Build Segments

Before building and loading a DB2 Server for VM segment, you must obtain the following information:

- Determine the beginning and ending storage page ranges for each segment you will be defining. Use one of the following tables to determine the size of the segment. Table 17 shows the sizes of the components that can be defined in saved segments that includes the base code, or the base code and Data Spaces Support. Table 18 shows the sizes of the components that can be defined in saved segments that includes the base code with DRDA code.

Table 17. Sizes for Saved Segments (Base Code)

<i>segname</i>	Segment Size in Kilobytes	DEFSEG Number of Pages (Segment Size / 4 Kilobytes)
SQLRMGR	320	80
LANGS001	320	80
SQLISQL	384	96
SQLSQLDS	1536	384
SQLXRDS	2112	528

Table 18. Sizes for Saved Segments (DRDA)

<i>segname</i>	Segment Size in Kilobytes	DEFSEG Number of Pages (Segment Size / 4 Kilobytes)
SQLRMGR	2304	576
LANGS001	320	80
SQLISQL	384	96
SQLSQLDS	1536	384
SQLXRDS	3968	992

The address where a segment can be defined on your system is dependent on the locations of the other saved segments on your system. See your system programmer for assistance in determining origin values for the saved segments.

If you are going to save a segment in an address space where an old saved segment is located, you must first purge the old saved segment. See the *VM/ESA: Planning and Administration* manual for more information on defining and deleting segments with VMSES/E.

- If you need to specify a storage key value other than the default value, 13, for the saved segment, refer to the VMSES/E Considerations under “ARISAVES EXEC” on page 493 and continue to the following step. Otherwise, continue with the following step.

Step 2. Log On to the Installation User ID

Log on to the installation user ID, 5648A70S.

Step 3. Access VMSES/E Code and Software Inventory Minidisks

To link and access the VMSES/E code and Software Inventory minidisks, enter the following commands. (You need R/W access to the Software Inventory minidisks.)

```
access 5e5 b
link maint 51d 51d mr
access 51d d
```

Do these steps for each segment

Step 4. Prepare to Add DB2 Server for VM Segment Definitions

Enter the following command to display the Segment Map panel, which displays information about the segments defined on your system:

```
vmfsgmap segbld esasegs segblist
```



```

                                Add Segment Definition
                                Lines 1 to nn of nn

OBJNAME.....:  segname
DEFPARMS....:
SPACE.....:
TYPE.....:  SEG
OBJDESC....:
OBJINFO....:
GT_16MB....:  NO
DISKS.....:
SEGREQ.....:
PRODID.....:  5648A70S  compname
BLDPARMS...:  UNKNOWN

F1=Help      F2=Get Obj   F3=Exit      F4=Add Line  F5=Map       F6=Chk MEM
F7=Bkwd      F8=Fwd       F9=Retrieve   F10=Seginfo F11=Adj MEM  F12=Cancel
====>

```

Figure 65. Add Segment Definition Panel Example

Step 6. Obtain the DB2 Server for VM Segment Definitions

To obtain the DB2 Server for VM segment definitions, you must fill in the appropriate fields on the Add Segment Definition panel.

OBJNAME.....: *segname*

<u>segname</u>	<u>Description</u>
SQLRMGR	Resource adapter
SQLISQL	ISQL
SQLSQLDS	DBSS
SQLXRDS	RDS
LANGxxxx	DB2 Server for VM message repository, where xxxx is:
	S001 English (mixed case)
	S002 English (uppercase)
	S003 French
	S004 German
	D001 Japanese
	D003 Chinese_Simplified

PRODID.....: *prodid compname*

prodid is 5648A70S. (This is the *prodid* for the base, mixed case English, as well as all other NLS Languages.)

If you are building the NLS message repository segments, use the base *prodid* 5648A70S.

compname is DB2VM or DB2VMSFS. Use DB2VM for building segments from a minidisk. Use DB2VMSFS for building segments from SFS directories.

Press PF10 to obtain DB2 Server for VM segment information.

Notes:

1. If you are setting initial segment definitions, you will receive message VMFSMD2038E. This is OK. You will fill in the DEFPARMS field on the next panel.
2. If the segment is already defined, you will receive the following message:
VMFSMD2044W Segment name *segname* already defined. Current segment definition will be replaced.

You can change the name of the segment in the next step if you do not want to replace the current definition.

Step 7. Update the DB2 Server for VM Segment Definition

Fill in or update the Add Segment Definition panel.

```

                                     Add Segment Definition
                                     More: +
                                     Lines 1 to nn of nn

OBJNAME....:  segname
DEFPARMS....:  ????-???? SR
SPACE.....:
TYPE.....:    SEG
OBJDESC....:  object_description
OBJINFO....:  object_information
GT_16MB....:  NO|YES
DISKS.....:
SEGREQ.....:
PRODID.....:  5648A70S compname
BLDPARMS...:  PPF(5648A70S compnameSEG bldlist_name)

VMFSMD2760I SEGINFO processing completed SUCCESSFULLY

F1=Help      F2=Get Obj   F3=Exit      F4=Add Line  F5=Map       F6=Chk MEM
F7=Bkwd      F8=Fwd       F9=Retrieve  F10=Seginfo F11=Adj MEM  F12=Cancel
====>
```

Figure 66. Add Segment Definition Panel Showing the New Segment Information

OBJNAME....: *segname*

If you want to change the name of the segment, replace *segname* with the new name.

DEFPARMS....:

Fill in the beginning and ending page ranges you calculated in “Step 1. Plan to Define and Build Segments” on page 189.

SPACE.....: *spacename*

Specifying *spacename* allows you to take advantage of segment packing by putting more than one component in a single segment space. Only segments that are used together (for example, ISQL and the Resource Adapter) should be put in a segment space together. This is because the entire segment space is loaded when one of the segments it contains is loaded. The DBSS

segment must be below 16M, but the RDS segment is usually defined above 16M. Therefore, they cannot be in the same segment space. To place them in the same segment space, the RDS segment must be saved below 16M. It is highly recommended that this NOT be done and RDS be saved above 16M.

GT_16MB....:

Specify N0 unless you are defining segments for the Resource Adapter or RDS components, which can (and usually should) be defined above 16M.

You can define and use the Resource Adapter and RDS saved segments above 16MB. To define the component above 16MB, specify the starting and ending pages above 16MB on the DEFPARMS field. If you define the component both above and below 16MB, give them different names.

Note: If you define the RDS segment above 16M, you **CANNOT** start the server with the "AMODE(24)" initialization parameter.

BLDPARMS....:

If you have your own PPF override, you must change the BLDPARMS field to reflect this.

Notice that the component name used in this field is DB2VMSEG or DB2VMSFSSEG. If you have a PPF override to the DB2VM or DB2VMSFS component name, you will also need to add an override to your PPF for DB2VMSEG or DB2VMSFSSEG before you build the segment in "Step 14. Build the DB2 Server for VM Segments" on page 197.

Step 8. Display Refreshed Segment Map Panel

Press F5 to display the refreshed Segment Map panel.

VMFSGMAP - Segment Map					More: - Lines nn to nn of nn	
		000-MB	001-MB	002-MB	003-MB	
Name	Typ	0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF				
M CMS	SYS	W-W-----1.....2.....3.....				
M GCS	SYS	W-----1.....2.....3.....				
		004-MB	005-MB	006-MB	007-MB	
Name	Typ	0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF				
CMSPIPES	DCS	4.....5.....6.....RRRRR-----				
M GCS	SYS	RRRRRRNNNNNNNNNNNNNNNNNNNN6.....7.....				
		008-MB	009-MB	00A-MB	00B-MB	
Name	Typ	0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF				
DOSBAM	SPA	8.....9.....-----				
CMSBAM	MEM	8.....9.....A.....RRRR.....				
CMSDOS	MEM	8.....9.....A.....R.....				
SQLISQL	MEM	RRRRR.....9.....A.....B.....				
CMSVMLIB	DCS	RRRRRRRRRRRRRRR9.....A.....WB.....RRRRRR				
F1=Help	F2=Chk Obj	F3=Exit	F4=Chg Obj	F5=File	F6=Save	
F7=Bkwd	F8=Fwd	F9=Retrieve	F10=Add Obj	F11=Del Obj	F12=Cancel	
====>						

Figure 67. Segment Map Panel with Added Segments Example

Step 9. Save the New Segment Information

Press F5 to save the changed information and exit from the panel.

_____ End of Do these steps for each segment _____

Step 10. Verify Virtual Storage

To load the saved segment, the machine must have enough virtual storage to contain the saved segment, loader tables, and CMS control block tables at the end of virtual storage.

Step 11. Prepare to Build the DB2 Server for VM Segments

Before building the new DB2 Server for VM segment, following these steps:

1. Clear your virtual machine by entering the following IPL command. This command bypasses loading the installation saved segment (CMSINST) and bypasses executing the System Profile EXEC.

```
ipl cms parm clear nosprof instseg no
```

Note: ** DO NOT press ENTER at the VM READ!**

2. Bypass the execution of the PROFILE EXEC by entering the following command:

```
access (nosprof
```

3. Access the VMSES/E code by entering the following command:

```
access 5e5 b
```

4. Link and access the Software Inventory disk by entering the following commands:

```
link MAINT 51d 51d mr  
access 51d d
```

5. Access the database machine, SQLMACH, production minidisk or SFS directory by entering the following command:

```
access vdev k
```

vdev is the address the database machine production minidisk is linked as by the installation user ID, or *vdev* is the name of the database machine production SFS directory. You need write access to this minidisk or directory.

6. Before running the ARISAVES EXEC to save the segments, activate the user language files by entering the following CMS command:

```
set language ameng (add ari user
```

ARISAVES is called by the VMFBLD command, in "Step 14. Build the DB2 Server for VM Segments" on page 197.

Step 12. Update the ARISSEGC Macro

Before you run the ARISAVES EXEC for each component to be defined in a saved segment, verify that the ARISSEGC macro contains the proper SYSNAME entry, origin, and saved segment choice for the component. Use XEDIT to change the ARISSEGC macro on the database machine, SQLMACH, production minidisk or SFS directory. (You should not use the VMSES/E local modification procedure.)

If the saved segment you defined is to become the new default, the SYSNAME value in the ARISSEGC macro should match the name you used to define the saved segment for the component. If not, edit the ARISSEGC macro and change the SYSNAME accordingly. The message ARI0365W will be issued if the SYSNAMEs do not match.

The ARISSEGC MACRO has a record length of 80, and its record format is fixed. The values in each record must be separated by one or more blanks and are interpreted in the order shown in the following list:

compid is the component ID as specified in the ARISAVES EXEC. Values of *compid* are:

- DBSS
- RDS
- ISQL
- RA

Yes|No indicates whether a saved segment should be used for this component. Both Y and YES indicate that a saved segment is used for this component. Both N and NO indicate that a saved segment is not used for this component.

origin is the hexadecimal load address of the saved segment, as specified in the DEFPARMS field of the Add Segment Definition panel.

segname is the name of the saved segment as defined in the OBJNAME field of the Add Segment Definition panel.

ARISAVES EXEC only processes the first occurrence of the value of each *compid*. Other records are ignored. You could use them to store other information on saved segments.

Unless you have changed some values in it, the ARISSEGC MACRO contains the following information:

```

* ARISSEGC MACRO - Saved Segment Control File
*
* 1. Change the values in the columns at the bottom of this file
* by overtyping them. Descriptions of the columns and possible
* values are:
*
* Column                Values
* -----
* COMPONENT ID          - RA, ISQL, DBSS, RDS.
* SAVED SEG             - Y, YES, N, NO. Answer YES to use saved
*                       segments for this component. Answer
*                       NO to load this component into user
*                       free storage.
* ORIGIN                - hexadecimal starting location of
*                       components in save segments.
*                       - no meaning if component is loaded into
*                       user free storage.
*                       - must start from column 35.
*                       - can be 6 to 8 digits long.
* SYSNAME               - As defined in the DEFSEG command.
*
* 2. Enter "FILE" on the command line to continue processing and save
* your changes.
*
* 3. You can include comments in this file. Place them at
* the end of the file and make sure that the first column
* contains an asterisk(*).
*
*COMPONENT            SAVED SEG      ORIGIN            SYSNAME
*
RA                    NO                ???????          SQLRMGR
ISQL                  NO                ???????          SQLISQL
DBSS                  NO                ???????          SQLSQLDS
RDS                   NO                ???????          SQLXRDS
END ARISSEGC MACRO <--- THE REQUIRED LAST ENTRY IN ARISSEGC MACRO

```

Figure 68. ARISSEGC MACRO

Step 13. Release the Production Minidisk or SFS Directory

Release the database machine, SQLMACH, production minidisk or SFS directory by entering the following command:

```
release k
```

Step 14. Build the DB2 Server for VM Segments

For each segment that is to be built, enter the following command:

```
vmfbld ppf segbld esasegs segblst segname (serviced
```

<u>segname</u>	<u>Description</u>
SQLRMGR	Resource adapter
SQLISQL	ISQL
SQLSQLDS	DBSS
SQLXRDS	RDS
LANGxxxx	DB2 Server for VM message repository, where xxxx is:

S001	English (mixed case)
S002	English (uppercase)
S003	French
S004	German
D001	Japanese
D003	Chinese_Simplified

yourname name specified on the Add Segment Definition panel in “Step 7. Update the DB2 Server for VM Segment Definition” on page 193.

You will be prompted asking you if you want the saved segments to be the new default saved segments. If you reply YES, ARISAVES updates the origin values in the ARISSEGC MACRO and generates bootstrap modules by calling the SQLBOOTS EXEC.

If you reply NO, you must create a bootstrap package yourself by using the SQLGENLD EXEC shown in “Step 15. Create a Bootstrap Package.”

Step 15. Create a Bootstrap Package

If you responded YES when prompted by the ARISAVES EXEC to use the saved segments you that loaded as defaults, you do not have to do this step, as ARISAVES would have generated a default bootstrap package (SQLDBA) for you.

If you answered NO to the prompt, you must run the SQLGENLD EXEC to create a bootstrap package for the saved segments you loaded. To run SQLGENLD EXEC, you must log off of the installation user ID and log on to the database machine (SQLMACH).

Because SQLGENLD prompts you for certain information about the new bootstrap, you should determine the contents of the bootstrap package before you run the SQLGENLD EXEC. For more information, see “Contents of a Bootstrap Package.”

Contents of a Bootstrap Package: A bootstrap package contains modules created by the SQLGENLD EXEC. SQLGENLD places the modules on the production minidisk (Q-disk). Note that, even though the DBSS and RDS components are loaded in different saved segments, there is only one bootstrap module for them. All of those components are needed to run the DB2 Server for VM system code in a database machine. Thus, one bootstrap identifies the location of the DBSS and RDS components.

Not all modules are needed because the database manager uses defaults when a module of a bootstrap is missing. For more information on the defaults, see “Using SQLGENLD” on page 199.

Figure 69 summarizes the different bootstrap modules that you can have.

	<u>fn</u>	<u>ft</u>	<u>fm</u>
Resource adapter -->	<i>dcssid</i>	SQLRMBT	Q
DBSS/RDS----->	<i>dcssid</i>	SQLDBBT	Q
ISQL ----->	<i>dcssid</i>	SQLISBT	Q

Figure 69. Bootstrap Package Contents

The *dcssid* (saved segment ID) is the name you give to the bootstrap package with SQLGENLD. It is the *dcssid* that you use in the DCSSID parameter of various IBM-supplied execs (such as, SQLSTART or SQLINIT). When *dcssid* is specified in a DCSSID parameter, the bootstrap package production disk entries are copied to the execution machine's A-disk as shown in Figure 70.

Production Q-disk Entry			COPY/RENAME	Execution Machine A-disk Entry		
FN	FT	FM		FN	FT	FM
<i>dcssid</i>	SQLRMBT	Q	TO	ARISRMBT	MODULE	A
<i>dcssid</i>	SQLDBBT	Q	TO	ARISDBBT	MODULE	A
<i>dcssid</i>	SQLISBT	Q	TO	ARISISBT	MODULE	A

Figure 70. Bootstraps Copied to the Execution Machine A-disk

The resource adapter bootstrap is incomplete when it is copied to the A-disk of the user machine. It is completed when the user runs the SQLINIT EXEC, which supplies the missing server name to be accessed.

Use SQLGENLD to generate bootstrap packages for running the database manager in saved segments. You cannot use this EXEC to generate a bootstrap package for running the database manager in a default mode. The SQLDBA bootstrap package identifies the default mode, which can be default saved segments (if you have defined them) or user free storage.

Using SQLGENLD: When you identify the bootstraps to be contained in the package you are creating and the location where you want them to load the code, you can use the SQLGENLD EXEC. To use SQLGENLD, obtain read access to the service minidisk by entering the following command:

```
access 193 v
```

You can run SQLGENLD only from the database machine:

```
sqlgenld
```

When it runs, the SQLGENLD EXEC obtains both read and write access to the production minidisk. Both kinds of access are available to a defined database machine. You should ensure that no other machine has write access to the production minidisk when you run SQLGENLD.

If you are running SQLGENLD from a database machine that does not own the production minidisk, SQLGENLD prompts you for the write password.

The SQLGENLD EXEC prompts you for *dcssid*. This is the name of the new bootstrap package. If a bootstrap package with this name already exists, SQLGENLD replaces the existing bootstraps. The EXEC does not let you replace the initial SQLDBA bootstrap package. The SQLDBA bootstrap package is used as a default by many IBM-supplied execs. Do not modify or erase the SQLDBA bootstrap package.

When you supply *dcssid*, SQLGENLD prompts if you want to create a resource adapter bootstrap, a DBSS/RDS bootstrap, and an ISQL bootstrap. For each bootstrap that you choose to create, you are prompted for the saved segment

name (or, in the case of DBSS/RDS, names). The name is the name you used in the OBJNAME field of the Add Segment Definition panel.

The database manager prompts if you want this bootstrap package to be the default DCSSID for user machines that have a link to this production (Q) disk. Specify this as the default if you have users linking to this Q-disk who will be accessing a database machine that does not own this production (Q) disk, and if you do not have saved segments identified by the SQLDBA bootstrap package. Because the database manager provides a default DCSSID, these users are not required to specify the DCSSID parameter when they run the SQLINIT EXEC.

Note: The SQLDCSID DEFAULT file cannot be used by a user if the file *resid* SQLDBN exists on the production (Q) disk they are linked to. This is because the default bootstrap package for a database is identified in the *resid* SQLDBN file. The SQLDCSID DEFAULT file is used by users that are accessing an application server other than the one that owns the Q-disk to which they are linked.

If you say that you want this bootstrap to be the default for users with a link to this production (Q) disk, a new file SQLDCSID DEFAULT will be created on the production (Q) disk to contain the default DCSSID. When the bootstraps are created, SQLGENLD places them on the production minidisk. They are then erased from the database machine A-disk.

Step 16. Verify the Virtual Storage Size

To run the database machine, you must ensure that its virtual storage is smaller than the load address of any of the saved segments that you loaded, or that the SEGMENT RESERVE command has been issued on the database machine for all segments that reside within the database machine virtual storage. The SEGMENT RESERVE commands should normally be placed in the Server's PROFILE EXEC.

Running in User Free Storage after Using Default Saved Segments

If you are using default saved segments and you want to run a component in user free storage, follow these steps:

1. Edit the production (Q-disk) copy of the ARISSEGC MACRO. Change the Y or YES to an N or NO for the components that you want to run in user free storage.
2. Run the SQLBOOTS EXEC for the components.
3. Ensure that you are using the SQLDBA bootstrap package. On the database machine, specify DCSSID(SQLDBA) on the SQLSTART command. On the requester, specify DCSSID(SQLDBA) when you invoke the SQLINIT EXEC.

If you have secondary production disks, you must manually copy the ARISSEGC MACRO to them. Then, you must run the SQLBOOTS EXEC for each production disk. For information on the SQLBOOTS EXEC, see "SQLBOOTS EXEC" on page 496.

ARISNLSC MACRO

The ARISNLSC MACRO indicates the repository used for DB2 Server for VM messages. It has a record length of 80, and a fixed record format. The MACRO is shown in Figure 71.

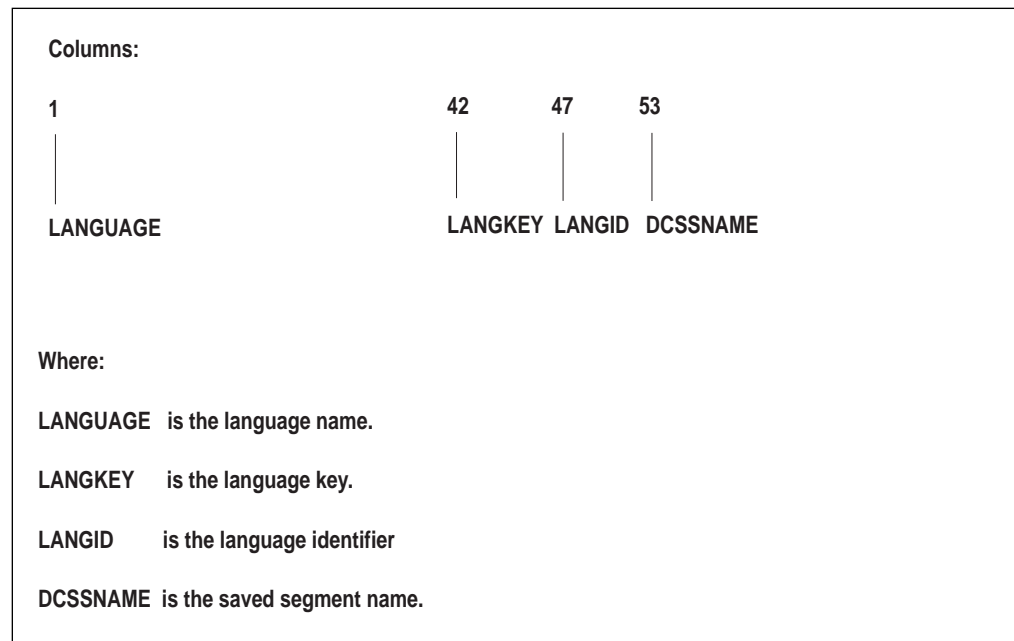


Figure 71. The ARISNLSC MACRO

For more information, see “National Language Support for Messages and HELP Text” on page 348.

Chapter 9. Making Backups and Recovering from Failures

Database recovery refers to the processing done to correct data when something goes wrong. This chapter presents a detailed description of basic recovery concepts, and how to implement them. More advanced recovery topics are discussed in Chapter 10, “Special Topics in Recovery Design” on page 237.

The problems that can occur fall into four categories:

Application Error

Occurs when an application (for example, an ISQL command or routine, or the DBS utility) does not end successfully.

User Logic Error

Occurs when the system or application does the requested function, but the request itself is in error — that is, the user (or application program) did not specify the correct function. For example, the user may have accidentally dropped the wrong table or dbspace.

This is the only type of error where detection is not immediate. Therefore, it presents more of a problem. Errors in the data can go undetected for quite some time, making recovery processing very complex.

System Failure

Occurs when the application server ends abnormally. Such failures can occur because of a severe error involving the operating system, or because of certain error conditions detected by the database manager, such as a power failure.

DASD Failure and Database Corruption

Occurs when the database manager cannot read data from or write it to the DASD where it is stored, because the storage medium is unreadable or damaged. Such an error (also called a media failure) can occur on the log, the directory, or a data extent (DBEXTENT).

This manual discusses how to recover from system and DASD failures. Recovery from application and user logic errors is described in the *DB2 Server for VM Database Administration* manual.

There are two aspects to dealing with system and DASD failures:

- Establishing and maintaining regular recovery procedures, to ensure that you have the information available to correct the data if something goes wrong.
- Correcting the data.

Understanding Recovery Concepts

To effectively protect your data and recover it in the event of failure, you need to understand the measures built into this product. Protecting against system failures involves the *LUW*, the *log*, and the *checkpoint*. Protecting against DASD failures entails two types of archive: the *database archive* and the *log archive*.

What is a Logical Unit of Work?

The data in your database is in a *consistent* state if no changes are left only partially completed.

Some data changes cannot be expressed in only one SQL statement. For example, suppose you have a banking program to transfer money between accounts, and want to transfer \$100 from a SAVINGS to a CHECKING account. The program makes this transfer in two steps:

1. Add \$100 to the balance of the CHECKING account.
2. Subtract \$100 from the balance of the SAVINGS account.

If the second step fails (for example, because of a system failure), the data is in an *inconsistent* state. That is, a deposit has been made to the CHECKING account, but no withdrawal has been made from the SAVINGS account.

The *logical unit of work (LUW)* prevents such inconsistencies. An LUW is a sequence of SQL statements that the system treats as a single entity. Either all the data changes made during an LUW are performed, or none is performed. In the example above, the two updates should be placed within a single LUW.

To group several SQL statements into one LUW, one uses the COMMIT WORK and ROLLBACK WORK commands.

If no problems or errors occur, the user issues the COMMIT WORK command to save all the changes made. If a problem occurs in the middle of an LUW, the user can issue the ROLLBACK WORK command to undo all the changes made since the last COMMIT WORK command.

An LUW can be as small as one SQL statement, or as large as an entire ISQL session or application execution. ISQL, by default, treats each command as an LUW, and issues a COMMIT WORK command after each SQL statement that modifies the database. Users can change this default by issuing the SET AUTOCOMMIT OFF command. For more information on the use of the AUTOCOMMIT, COMMIT, and ROLLBACK commands, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

What is a Log?

The log is a file maintained on DASD that records all the changes completed by each LUW. For each change, the log records the old and new values of the updated object. If any changes to the database must be undone or redone, you can use the log to restore the data to its proper state.

In addition to the changes made by each logical unit of work, the log also records when each logical unit of work started and stopped. (It does not record logical units of work that only read information from the database).

A database must have at least one log. Optionally, you can define a second log. If there are two logs, they are exact duplicates: then, if a DASD failure occurs on one log, the database manager can continue using the other copy. For more information, see "Using Dual Logging" on page 239.

Larger logs may be needed for tables that are being captured for DataPropagator because of the increased amount of log data written for UPDATES to those tables

which specify DATA CAPTURE CHANGES. Tables being captured will log the entire original row (not just the data that was changed), and the new data that replaces the old changed data. You should consider increasing the size of the log dbextent(s) when planning to make extensive use of this function.

What is a Checkpoint?

Checkpoints are taken periodically. During a checkpoint the database manager stops servicing users, and takes a “snapshot” of the database that includes updates from completed LUWs as well as from those that are still in progress, and writes them to DASD. In addition, a special checkpoint record is written to the log to synchronize the log with the state of the database.

What Happens after a System Failure?

Restart Recovery with a Log

If your system fails, as long as the current log is available, the database will be automatically recovered to a consistent state when you restart the application server. This process, called *restart recovery*, uses the log to ensure that changes made by LUWs are either committed (if they had successfully finished) or backed out (if they had not finished successfully).

The recovery process determines the state of each LUW; both at the time of failure and at the time of the last checkpoint before the failure. The following scenarios are shown in Figure 72 on page 206:

- LUW A: if the LUW starts and ends before the checkpoint, all the updates are safely reflected in the database at the checkpoint.
- LUW B: if the LUW starts before the checkpoint and commits work after the checkpoint but before the failure, those updates made after the checkpoint must be redone, using the log. Those updates made prior to the checkpoint are reflected in the database.
- LUW C: if the LUW starts before the checkpoint but is not completed before the failure, those updates made prior to the checkpoint must be undone using the log. The updates made after the checkpoint are not reflected in the database: thus all the updates must be re-entered.
- LUW D: if the LUW starts after the checkpoint and commits work before the failure, all its updates must be redone using the log.
- LUW E: if the LUW starts after the checkpoint and is not completed before the failure, all its updates must be re-entered since none of them are reflected in the database.

The following diagram illustrates the LUW Recovery process for the five cases described above:

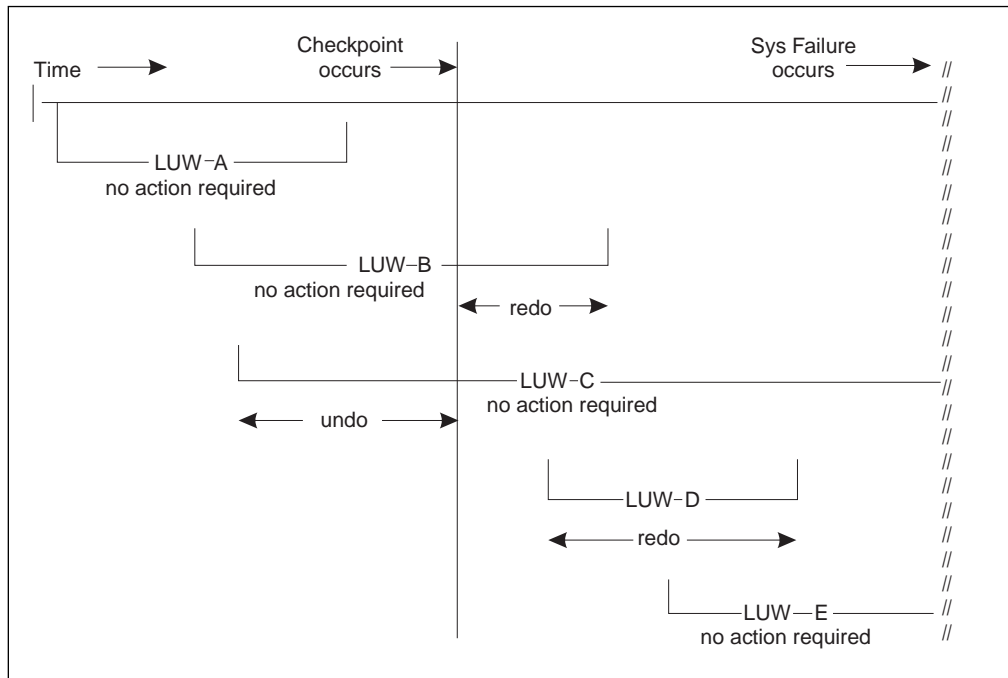


Figure 72. LUW Recovery Actions

Restart Recovery Without a Log

If the application server must be restarted without a log (due to the log either being lost, reformatted, or reconfigured immediately after the failure), the database cannot be adjusted to complete committed logical units of work or to back out uncommitted ones. In this situation, to recover the database you will have to restore a previous database archive, together with any applicable log archives.

If the database manager had been running in single user mode with LOGMODE=N, the changes made by the application are not logged. However, a checkpoint would have been taken each time the application issued a COMMIT WORK (or one was issued for the application), so most changes will have been effectively committed. Any that were uncommitted at the time of failure will be discarded when you restart the application server and will need to be re-entered.

What is an Archive?

Archiving facilities enable you to recover your database directory and storage pools from DASD failures. There are two kinds of archives: database archives and log archives.

Database Archives

A *database archive* is a tape copy of the database directory and dbextents. It can be taken using two types of facilities:

- *database manager* archiving facilities supplied with this product
- *user* archiving facilities such as the VMBACKUP management system or the BACKUP command of the Data Restore Feature.

If database manager facilities are used, the database manager takes a checkpoint (the begin-archive checkpoint) and writes a copy of the database directory and the database to tape, as they were at the checkpoint. (A database archive does not

include a copy of the log.) Users continue to receive service while the archive is being done.

A user archive can only be done while the application server is shut down. A user archive generally takes less time than a database manager archive.

You are not restricted to using one kind of archive for a given database; you can switch between database manager archives and user archives as often as you like. There are two situations in which the former facility is required:

- When you migrate a database between two different operating systems (for example, from VSE to VM)
- When a database archive is needed while users are accessing the database. You can avoid this situation by using log archiving (LOGMODE=L).

Experience helps you determine which method is best for you. When using any backup method, the performance improvement will be related to how full your database is. The fewer pages in your database that are allocated, the less time a database manager archive takes.

In fact, if the percent of allocated pages is low enough, a database manager archive will outperform a user archive, because the database manager only archives pages that actually contain data. User facilities archive all pages, so the time taken does not vary with the number of pages allocated.

Aside from the performance advantage that user archiving facilities may offer because they exploit particular device characteristics, consider whether your facility provides other advantages such as archiving multiple dbextents simultaneously.

For a description of how to carry out these archives, see “Performing Database Archives With Database Manager Facilities” on page 211 and “Performing Database Archives With User Facilities” on page 215.

Log Archives

A *log archive* is a copy of the log on tape or disk. Only database manager archive facilities can be used to archive the log. Log archives can be taken either when the database manager is running or at shutdown. Because the log is usually much smaller than the database, this archive takes less time than a full database archive. For a description of how to carry it out, refer to “Performing Log Archives” on page 216.

Recovering from DASD Failures that Damage the Database

If a DASD failure occurs on one of your database devices, you can restore the database by replacing the damaged minidisk with a working minidisk (see “Replacing a Database Minidisk” on page 232), and then restoring the data from the archived database and logs (if applicable).

There are two ways to do this. One way is to use the database archive and the current log. By loading the archive and re-applying the changes in the log, you can bring the database up-to-date because all changes made to the database since the archive are recorded in the current log. If the restore set for the database archive includes the current log, you can recover the damaged storage pools instead of the entire database using the Data Restore Feature. See the *DB2 Server for VSE & VM Data Restore* manual for more information on storage pool level recovery.

Alternatively, if you archived the log, you can use the database archive, the log archives you created since the last database archive, and your current log, to recreate the database. You would load the database archive, and reapply the changes in the log archives and the current log. If the restore set for the database archive includes the current log, you can recover the damaged storage pools instead of the entire database using the Data Restore Feature. See the *DB2 Server for VSE & VM Data Restore* manual for more information on storage pool level recovery.

The relationships among the different archives, the current log, and the current database are shown in Figure 73 on page 210. For more details, see “Restoring the Database” on page 224.

Recovering from DASD Failures that Damage a Log

If a DASD failure occurs, such as an unresolvable I/O error, on one of the log devices, there are two possibilities for recovery:

1. If you are single logging, replace the damaged log minidisk (see “Replacing a Log Minidisk” on page 235), and then follow the procedures in “Log Reconfiguration” on page 240. Log data from the damaged log is lost.
2. If you are dual logging, replace the damaged log minidisk with a working minidisk (see “Replacing a Log Minidisk” on page 235), and then start the application server with the same log mode used before the log minidisk was damaged. The contents of the good log minidisk is copied to the new log minidisk.

Recovering from DASD Failures that Damage the Database and Log

If a DASD failure occurs on both a database device and a log device, you can restore the database by replacing the damaged database minidisk with a working minidisk (see “Replacing a Database Minidisk” on page 232), replacing the damaged log minidisk with a working minidisk (see “Replacing a Log Minidisk” on page 235), and then restoring the data from the archived database and logs (if applicable) as described in “Restoring the Database” on page 224.

Establishing DASD Recovery Procedures

As the system administrator, you must establish recovery procedures for your installation. The procedures you put in place will determine the degree of protection for your database. Naturally, trade-offs exist; when you allocate system resources to protect against failures, these resources are unavailable to other users. However, if a failure occurs, the recovery takes less time.

This section discusses some of the options available. Based on this information, devise a plan that best suits your requirements.

Choosing a Log Mode

One of the first decisions you must make when designing a recovery strategy is the type of *log mode* you want. The log mode is an initialization parameter that you specify when you start the application server. It has four possible values:

`LOGMODE=Y`

All changes to the database will be recorded in a log, but no archives of the log or database will be maintained. This value is the default. Use it if

you do not need to protect your data from DASD failures. The application server will run faster, since it will not require the extra time to create archives.

LOGMODE=A

All changes to the database will be recorded in a log, and regular archives of the database will be maintained. You can either create these archives yourself, or have them created automatically when the log reaches a certain threshold level.

LOGMODE=L

All changes to the database will be recorded in a log, and regular archives of the log will be maintained. You can either create these log archives yourself, or have them created automatically when the log reaches a certain threshold level (to prevent it from becoming too full to be effective).

Log archives do not contain data, but only operations that change the database. If you use this log mode, you must take an occasional database archive as well. If a failure occurs, you can use the database archive, subsequent log archives, and the current log to recover the database.

The log archives must be continuous, recording all processing that occurred since the last database or log archive. If a gap exists, it will be impossible to restore the database to its current level. (The processing that occurred during the gap can never be reapplied to the database because it was never archived.) Gaps can occur in the sequence of log archives when, for example, you switch from LOGMODE=L to some other log mode. If the continuity of the log is broken in this manner, the database manager will force a database archive before you return to LOGMODE=L processing.

LOGMODE=N

No changes to the database are recorded. This option, which is only available in single user mode, is not recommended for normal operation but can be useful in some situations. For example, it may be more efficient not to log changes if you are loading a large amount of data into a table by using the DBS utility in single user mode. If a problem occurs while you are loading, you do not need the log to recover; you can simply start over.

Once you have decided on a log mode, use it whenever you start the application server. **Do not change it without thought and planning.** If you must do so, you may have to carry out additional procedures. For information, refer to “Switching Log Modes” on page 237.

Deciding between LOGMODE=A or L

Figure 73 on page 210 illustrates the relationships among the archives, the log, and the database when the log mode is A or L. You should consider several things before choosing one mode over the other.

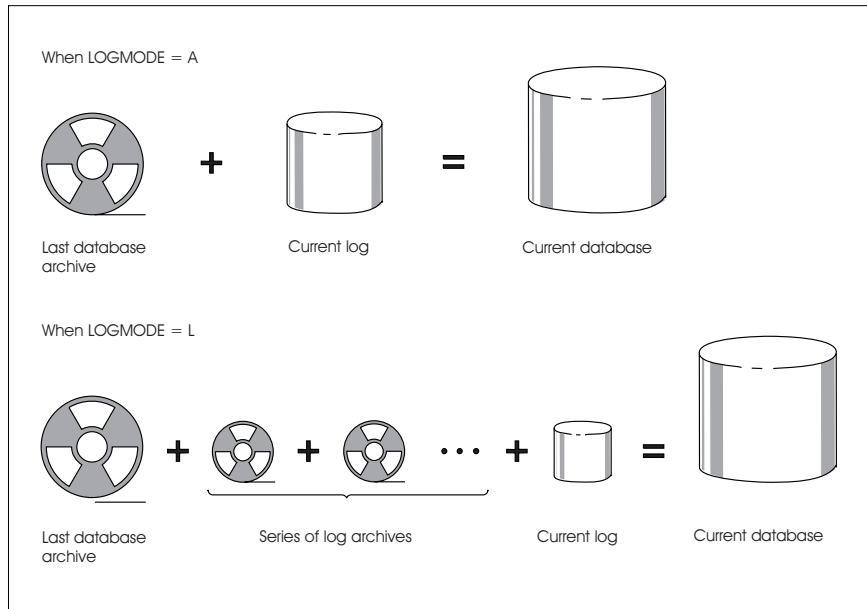


Figure 73. Relationships among the Archives, the Log, and the Database

There are two advantages to log archiving (LOGMODE=L):

- It usually takes less time, because only the log is being archived, not the directory and dbextents. This is especially helpful when the archive is being done to free log space when the database manager is running.
- If the last database archive is unreadable or unavailable, you can bring the database back to its current status by using a back-level database manager archive or user archive, and applying to it the changes that were recorded in all subsequent logs. More recent database archives are ignored when you restore a back-level database. Two requirements must be met in order for you to use this method:
 - The log archives must be continuous. That is, you cannot have switched log modes and done a COLDLOG (with SQLLOG) or a restore since the back-level archive was created.

Note: You can switch from LOGMODE=L to A and then back again without breaking the continuity of the log archives, provided that no database archive was taken while LOGMODE was set to A.
 - You have not added dbspaces, added or deleted dbextents, or reconfigured the logs since the back-level database archive was made. These operations are recorded in the database directory, so if you have carried any of them out, the directory will not be synchronized with the database changes.

A disadvantage of archiving the logs is that no logical units of work can be active during the checkpoint that immediately precedes the archive itself. Concurrent access is allowed once the checkpoint is complete, but users may experience delays both before and during the checkpoint.

Another disadvantage is that it takes longer to restore the database. For example, suppose you have been taking a database archive every Friday evening and a log archive on Tuesdays and Thursdays, and on a Friday afternoon there is a media failure on the DASD that contains the database directory. You must restore the

most recent database archive (from the previous Friday), and then restore the log archives from Tuesday and Thursday as well as the changes recorded in the log that was current at the time of the failure. Because only the changes to the database are stored in the log, restoring the database is similar, in processing time, to redoing all the work from the week. If there was heavy activity that week, restoration can take a long time.

Had you used database archives (LOGMODE=A) as intermediate online archives, you would only need to restore Thursday's database archive and reapply the changes on the current log. The restore time is much shorter. On the other hand, more time would have been spent doing the intermediate archives. Because media failures are infrequent, it is usually better to take intermediate log archives instead of intermediate database archives. Depending on your own experience with media failures, it may even be worthwhile to lengthen the time between database archives taken at shutdown.

Backing Up the History Area

The database manager uses the history area of the current log to keep track of recovery events (for example, database archives and log archives). The database manager can then determine which log archives belong with which database archives. The ARIHSDS ARCHIVE file contains the updated history area. If both the work disk, containing the ARIHSDS ARCHIVE file, and the disk containing the current log are damaged or are unavailable (offsite disaster recovery scenarios), you cannot use log archives to restore the database. To be able to apply log archives, you should create a backup file of the ARIHSDS ARCHIVE file after each log archive. You can then use this backup file to rebuild the log history area.

Archiving Procedures

This section describes how to create archives to protect your database against system failure. If a system failure occurs while you are taking an archive, see "Restarting from a System Failure While Archiving" on page 230.

Performing Database Archives With Database Manager Facilities

Database archives are tape copies of the directory and dbextents that are carried out using the database manager archiving facilities:

- By issuing an SQLEND ARCHIVE operator command, which copies the database to tape only after all LUWs complete. The copy contains all changes made by completed LUWs because no LUWs are active when the database archive is made. Log space is freed after the archive completes successfully. No changes made by incomplete LUWs are in the database archive copy. This method is preferred.
- By issuing an ARCHIVE operator command, which lets the operator initiate a database archive at any time without either shutting down the application server or stopping access to it. The drawback, however, is that if the archive is started while applications are accessing the database, the archive copy may contain changes made by incomplete LUWs, and cannot be used for recovery from user logic errors, unless the log that was current when the database archive was taken is available. For more information about user logic errors, see the *DB2 Server for VM Database Administration* manual.

The ARCHIVE command should be used only when you need to take a database archive to free log space but cannot afford to shut down the application server. Thus, you might want to schedule an SQLEND ARCHIVE for every Friday night, and periodic online archives during the week.

Log space used by completed logical units of work is freed. Log space reflecting changes that are not completely included in the database archive (as of its begin-archive checkpoint) cannot be reused until the next database archive that completely includes the changes.

- By reaching the ARCHPCT value, in which case a database archive is taken automatically. The ARCHPCT initialization parameter protects the log from overflowing. (See “ARCHPCT” on page 74.) When you are running the database manager with only database archiving active (LOGMODE=A), log space that can be freed by the archive is determined by the begin-archive checkpoint and freed by the end-archive checkpoint. Log space that has been used since the longest running active logical unit of work began cannot be reused until the next database archive is taken. If the log becomes filled to the ARCHPCT value, the database manager forces an online database archive.

Set the ARCHPCT value lower than the SLOGCUSH value, which determines when the log overflow procedure is started. When the log is filled to the percentage indicated in SLOGCUSH, the LUW that was running the longest is backed out. (Although this procedure allows the log space to be reclaimed by another forced online database archive, it can frustrate the user whose application was almost finished.)

Ideally, your log should be large enough so that the ARCHPCT value is never reached. If this value were reached at an inconvenient time (say when the operator is not at the console), database activity could stop. To prevent this from happening, you should use the ARCHIVE command to do online database archives when activity on the system is low.

Also, if you do have a database archive taken because ARCHPCT is reached, remember you cannot use this archive to recover from user logic errors. Like an online database archive initiated with the ARCHIVE command, it contains changes from incomplete LUWs, so you still need the log if this archive is the source for a restore.

Contention During an Archive

When a database archive is taken online, using database manager facilities only, other work usually continues. If, however, a condition arises during the archive that requires a checkpoint to be taken, other work must wait until the archive process completes. Such conditions include:

- A short-on-storage condition for a storage pool
- A full database log
- A COMMIT or ROLLBACK WORK statement issued during an LUW that updated data in a nonrecoverable storage pool.

Note: You can use the SHOW LOG operator command to monitor available log space to assist you in scheduling database archives. See the *DB2 Server for VSE & VM Operation* manual for description of operator commands.

Example of an **SQLEND ARCHIVE**

If you intend to create database archives, specify a value for the LOGMODE parameter of A or L when you start the application server.

SQLEND ARCHIVE with LOGMODE=A

To use the SQLEND ARCHIVE command when LOGMODE=A:

1. Log on using the user ID and password for the database machine.
2. Attach and mount a labeled tape at virtual device address 181. Indicate that you are writing to the tape. See the VM system administrator for the procedures used at your installation to label, attach, and mount tapes.
3. To identify who is connected to the application server, type the following command and press ENTER:

```
SHOW USERS
```

You see a list of the users connected to the application server. Some of these users could be accessing data in the database.

4. Notify the current users that you want to stop the application server. Ask them to complete their work and sign off. If any user does not stop voluntarily, consider using the FORCE command to disconnect that user. Use this command with caution, however, as the uncommitted work of the users you disconnect is rolled back.
5. To start the archive process and stop the application server, type the following command and press ENTER:

```
SQLEND ARCHIVE DVERIFY
```

You should verify the directory whenever you create a database archive by specifying the DVERIFY parameter for the SQLEND ARCHIVE command. If you do not verify the directory, inconsistencies in the control information are recorded in the database archive. A subsequent restoration using that archive would fail. When you verify the directory, the system displays a message if it finds an error, and does not create the database archive.

6. While the database manager is creating the database archive, a number of messages are displayed at your display terminal. Among them is ARI0239I.

Write the information that this message provides on the external label of each tape reel or cartridge. Include the date, time, and type of archive (database). See "Labeling Your Archive Tapes" on page 223 for further instructions on how to use this information. During a restoration, the same information is provided, so you can easily verify that you are using the proper tape volumes.

7. The database manager prompts you for the virtual device address of your tape:

```
ARI0299A Ready archive output volume. Enter the CUU.
```

Type the following and press ENTER:

```
181
```

8. When the tape is full, the system prompts you to mount a second tape. After you mount the second tape, type the following and press ENTER:

```
READY
```

A delay can occur as the database is copied to tape. The system continues to request that you mount new tapes until the database archive is completed.

The number of tapes you are required to mount depends on the size of your database.

When the archive is completed and the database is copied, you see the following message:

```
ARI0292I ARCHIVE is completed.
```

Note: While the archive is being created you can still enter operator commands.

When the application server stops, you see the messages:

```
ARI0032I The database manager has terminated.  
ARI0043I Database manager return code is 0.
```

You have finished creating the database archive.

SQLEND ARCHIVE with LOGMODE=L

If you run the application server with LOGMODE=L, the SQLEND ARCHIVE command ensures that you have an unbroken sequence of log archives by creating the log archive (if there is information in the log) before the database archive. You use the SQLEND ARCHIVE command when LOGMODE=L in the same way you do when LOGMODE=A, except that two tapes are needed for the procedure shown here — one for the database archive and one for the log archive, if a log archive is to be taken. It is possible to use a single tape by creating the log archive on disk. For more detail, see “Log Archiving to Disk” on page 219.

Using the SQLEND ARCHIVE DVERIFY with LOGMODE=L allows you to create a valid log archive, if there is information in the log, even if the system finds an error in the directory. However, in this situation, a database archive is not created.

To use the SQLEND ARCHIVE command when LOGMODE=L, do the following:

1. Follow the first six steps for using the SQLEND ARCHIVE command when LOGMODE=A, beginning on page 213. The database manager first creates a log archive if there is information in the log. You see the message:

```
ARI0254I The database manager is initiating a log archive.  
          When the log archive is complete, the database  
          manager will process the database archive request.
```

2. Instruct the system to create the log archive on tape if a log archive is to be taken. Tape is the default medium for a log archive.

The following messages prompt you for the output medium:

```
ARI0252I   Medium:   tape 183  
ARI0246D The above information describes the log archive  
about to be done. Enter either:  
          CONTINUE   to proceed using the output medium  
                    indicated, or  
          CHANGE     to change this medium.
```

3. To use the default medium, type the following and press ENTER:

```
CONTINUE
```

If you reply CHANGE, you can direct the log archive to disk. For more details on the CHANGE option, see “Log Archiving to Disk” on page 219.

4. When the tape is full, the system prompts you to mount another tape. After you mount a new tape, type the following and press ENTER:

READY

A delay may occur while the database manager archives the log to tape. When the log is archived, you see the following message:

```
ARI0292I ARCHIVE is completed.
```

Note: While the log archive is being created, you can still enter operator commands.

5. The system now continues with the database archive. Attach and mount a second labeled tape at virtual device address 181. Indicate that you are writing to the tape. Use the procedures set up at your installation to label, attach, and mount tapes. This tape is used for the database archive.
6. Repeat steps 6, 7, and 8 from the LOGMODE=A example that begins on page 213 to complete the database archive.

Performing Database Archives With User Facilities

User archives are database archives (LOGMODE=A or L) done with user facilities such as the VMBACKUP Management System (VMBACKUP-MS) or the BACKUP command of the Data Restore Feature. User archives include the database directory and all minidisks, but **not the logs**.

Because database manager archiving facilities are DASD-independent, they do not take advantage of particular DASD characteristics to improve performance. Some user facilities exploit these characteristics, and can archive and restore your database more quickly in some situations.

Note: If you use VMBACKUP-MS to create user archives, you must specify that the database directory and dbextent minidisks are non-CMS minidisks.

To begin archiving your database with user facilities, stop the application server and issue:

```
SQLEND UARCHIVE
```

After all logical units of work have been finished, the database manager indicates in the log history that a user archive will be taken, then prompts the operator to take the archive, and ends. (If LOGMODE=L and the log contains information, it takes a log archive before ending.) When the application server ends, the operator should take the user archive. The next time the application server is started, it displays a message to confirm that the user archive was done.

Note: Confirmation of a successful user archive is required at the next startup. If the operator specifies a restore (STARTUP=R or U) the next time the application server is started, the system assumes that the user archive was *not* taken. If the system does not prompt the operator to confirm that a user archive was created, this means that the archive was not recognized (whether or not it was successful), and it must be repeated.

Note: Do not stop the server with SQLEND QUICK and then take a user archive because the user archive will not contain consistent data.

Freeing Log Space during a User Archive

Log space is freed after a successful user archive has been confirmed at the next startup. If you take user archives and it becomes necessary to free log space when the database manager is running, you must use either the log or database archiving facilities supplied with this product to free the log space.

For log archives, set LOGMODE=L when starting the application server, and for database archives, set LOGMODE=A. In both cases, this will ensure that database archives are automatically taken if the log fills to the ARCHPCT value. Or, if you prefer to schedule your online archives yourself, periodically issue the LARCHIVE command for log archives, or the ARCHIVE command for database archives.

Note: You can use the SHOW LOG operator command to monitor available log space to assist you in scheduling user archives.

Performing Log Archives

A log archive is a copy on tape or disk of all the active pages of the database log except for the last one, the log history area. To use log archiving, set LOGMODE to L. A log archive can only be performed with database manager facilities supplied with this product.

Log archives can be used with database archives taken with either database manager facilities or user facilities. Each sequence of log archives must be preceded by at least one database archive.

The log archive process can be started in the following ways:

- By issuing an SQLEND LARCHIVE operator command, which causes the database manager to copy the log to tape or disk when all LUWs are complete. Log space is freed after the archive completes successfully.

For a description of this process refer to “Example of an SQLEND LARCHIVE” on page 218.

- By issuing an LARCHIVE command when the database manager is running. This should be done when you need to take an archive to free log space but cannot afford to shut down the application server. For example, you may schedule an SQLEND ARCHIVE or SQLEND LARCHIVE for every Friday night, and schedule periodic online log archives during the week. Log space is freed after the archive completes successfully.
- By reaching the ARCHPCT value, in which case a log archive is taken automatically. The ARCHPCT initialization parameter protects the log from overflowing. See “ARCHPCT” on page 74. When you run the database manager with log archiving active (LOGMODE=L), log space after the begin-archive checkpoint cannot be reused until the next log archive is taken. If the log becomes filled to the ARCHPCT value, the database manager forces an online log archive. This archive cannot begin until all active logical units of work have been either committed or backed out.

Set the ARCHPCT value lower than the SLOGCUSH value, which determines when the log overflow procedure is run and thereby protects the log from overflowing. (see “SLOGCUSH” on page 74.) When the log is filled to the percentage indicated in SLOGCUSH, the LUW that was running the longest is backed out. (Although this procedure allows the log space to be reclaimed by the online log archive, it can also frustrate the user whose application almost completed.)

Because a log archive finishes faster than a database archive, it has less performance impact if it is done when the database manager is running. If log archives are occurring at inopportune times, however, you may want to periodically issue LARCHIVE when activity on the database manager is low. Be sure the log is large enough so the ARCHPCT limit is not reached before your scheduled log archive.

- By doing an explicit database archive while LOGMODE=L by issuing SQLEND ARCHIVE, SQLEND UARCHIVE, or ARCHIVE. Before archiving the database, the database manager does an implicit log archive (if information is in the log). Note that the database manager never does an implicit database archive.
- By restoring the database. This causes the database manager to do a log archive (if there is information in the current log) before beginning the database restore.

Contention During an Archive

When an online log archive is requested, the database manager allows any LUWs that are active to finish, but prevents any new ones from starting. A message is displayed that tells how many LUWs are active. When they are complete, the database manager takes a checkpoint and creates the log archive. During the checkpoint, access to the database is disabled and any users or applications that try to start a new LUW will be in a lock wait.

You can monitor the locking contention caused by the online log archive by using the SHOW operator commands from the database machine console. However, you cannot issue SHOW commands from ISQL to monitor the lock contention.

In most situations, only a slight delay occurs before the checkpoint is taken, but if there are long-running LUWs, it can be longer. In a worst-case scenario, a long-running LUW can delay the log archive checkpoint long enough so that the SLOGCUSH value is reached, and the database manager must roll back the longest-running LUW to free log space.

If you find that users are experiencing long delays because the database manager is trying to take a checkpoint, you can issue the SHOW operator commands to determine which user is delaying the start of the checkpoint, and then issue the FORCE command to end that user's LUW.

During the creation of the log archive, normal access to the database is usually resumed. If, however, a condition arises during the archive that requires a checkpoint to be taken, other work must wait until the archive process completes. Such conditions include:

- A short-on-storage condition for a storage pool
- A full database log
- A COMMIT or ROLLBACK WORK statement issued during an LUW that updated data in a nonrecoverable storage pool.

Note: You can use the SHOW LOG operator command to monitor available log space to assist you in scheduling log archives.

Example of an **SQLEND LARCHIVE**

If you intend to create log archives, specify LOGMODE=L when you start your application server. You can create your archive on tape, disk, or on a combination of these media. There are different ways to start the process. The preferred method is to issue an SQLEND LARCHIVE operator command, which instructs the application server to copy the log to tape or disk as it shuts down.

Log Archiving to Tape

To use the SQLEND LARCHIVE command to create a log archive on tape:

1. Log on to the database machine.
2. Attach and mount a labeled tape at virtual device address 183. Indicate that you are writing to the tape. Use the procedures set up at your installation to label, attach, and mount tapes.
3. Start the application server with LOGMODE=L.
4. Notify the current users that you want to stop the application server. Ask them to complete their work and sign off. If a user does not stop voluntarily, consider using the FORCE command to disconnect that user. Use this command with caution, however, as the uncommitted work of the users you disconnect is rolled back.

To display a list of the users currently connected to the application server, type the operator command:

```
SHOW USERS
```

5. Archive the log and stop the application server by typing the following operator command:

```
SQLEND LARCHIVE
```

6. You will see the following message:

```
ARI0239I External labeling of this archive is:
          Type:      log archive
          Timestamp: 12-09-92 14:41:00
ARI0252I   Medium:   tape 183
```

Write the information that this message provides on the external label of each tape reel or cartridge. Include the date, time, and type of archive (log). See "Labeling Your Archive Tapes" on page 223 for further instructions on how to use this information. During a restoration, the same information is provided, so you can easily verify that you are using the proper tape volumes.

7. The following message prompts you to accept or change the default storage medium (tape is the default medium for a log archive):

```
ARI0239I External labeling of this archive is:
          Type:      log archive
          Timestamp: 12-09-92 14:41:00
ARI0252I   Medium:   tape 183
ARI0246D The above information describes the log archive
about to be done. Enter either:
          CONTINUE  to proceed using the output medium
                   indicated, or
          CHANGE    to change this medium.
```

Type CONTINUE and press ENTER. (If you reply CHANGE, you can direct the log archive to disk. For more details, see Log Archiving to Disk.)

If the tape becomes full, you will be told to mount another. After you mount the new tape, type `READY` and press `ENTER`. You will be asked for more tapes until the log archive is completed. The number of tapes required depends on the size of your log.

When the log is completely archived, the following message is displayed:

```
ARI0292I Archive is completed.
```

Note: You can still enter operator commands while the archive is being created.

Log Archiving to Disk

Caution

Before you create log archives on disk, be aware that disks are exposed to user errors (erasing a file containing an archive), and the remote possibility of hardware problems such as head crashes.

To minimize the impact of hardware problems, ensure that all log archive disk files are physically located on disk volumes that are not used for the various database extents. You can back up the log archives yourself to achieve a higher level of recoverability.

The output disk can be either a standard CMS minidisk or a shared file system (SFS) directory. If it is a CMS minidisk, the database machine must access it in read or write mode before you start the application server. If it is an SFS directory, the database machine must access it before you start the application server.

You might experience delays if you use a remote shared file system (SFS) directory. If these delays are causing problems with archiving the log, use a lower `ARCHPCT` value when starting the application server.

The disk must contain enough space to hold the archived log. You must monitor the space and erase old log archives to free space if need be, as the database manager does not check that enough space exists. To calculate the size of the CMS minidisk or shared file system directory needed to contain the log archives, use this formula:

$$(\text{log disk size}) \times (\text{SLOGCUSH \%}) \times (\text{maximum number of log archives on this disk})$$

Where:

log disk size is the size of the log disk used in the units you are using (for example, 4-kilobyte blocks)

SLOGCUSH % is the value used for the `SLOGCUSH` parameter of the `SQLSTART EXEC`. The default for `SLOGCUSH` is 90%, so you would use the value `.9` if you did not specify `SLOGCUSH` when you started the application server.

maximum number of log archives on this disk

is the maximum number of log archives you will store on this CMS minidisk or shared file system (SFS) directory.

To use the `SQLEND LARCHIVE` command to create a log archive on disk:

1. Log on to the database machine.

2. Start the application server with LOGMODE=L.
3. Notify the current users that you want to stop the application server (refer to "Log Archiving to Tape" on page 218).
4. Archive the log and stop the application server by typing the operator command:

```
SQLEND LARCHIVE
```

5. You will see the following message:

```
ARI0239I External labeling of this archive is:
          Type:      log archive
          Timestamp: 12-09-92 14:41:00
ARI0252I   Medium:   tape 183
ARI0246D The above information describes the log archive
about to be done. Enter either:
          CONTINUE   to proceed using the output medium
                    indicated, or
          CHANGE     to change this medium.
```

Type CHANGE and press ENTER, or type CONTINUE if you want to direct the log archive to tape. You will see the following message:

```
ARI0263D To direct the log archive to tape, enter TAPE followed
          by the tape address (CUU) to be used.
          To direct the log archive to disk, enter DISK followed
          by the disk file's file name, file type, and file mode.
          If you chose DISK, the default file is:
          SQLMACH3 12099201 ??
```

6. Type the following and press ENTER:

```
DISK = = fm
```

where *fm* is the file mode of the disk to which you want to write the log archive and the two equal signs give you the default file name and file type chosen by the system. In this example, the default file name is SQLMACH3, and the default file type is 12099201. You can change these by supplying your own values.

7. Type CONTINUE and press ENTER. When the log is completely archived, the following message is displayed:

```
ARI0292I Archive is completed.
```

Note: You can still enter operator commands while the archive is being created.

Changing the Default Medium: You can avoid having to define your disk each time you issue an SQLEND LARCHIVE, by using a file definition. To change the default medium to disk and to archive your log:

1. Log on to the database machine.
2. Issue a FILEDEF command for your log archive file, ARILARC, before you start the application server. For example:

```
FILEDEF ARILARC DISK fn ft fm (PERM BLKSIZE 28672
```

where *fn*, *ft*, and *fm* are the file name, file type, and file mode of your choice. The file name and file type you specify will not be used by the system, since it uses defaults of its own. You must only specify them so that the command will have valid syntax. The file mode you specify will be used by the system to determine on which disk to write the archive.

Note: After a restore operation, the FILEDEF for ARILARC is inoperative. You must specify it again by either:

- Issuing the FILEDEF command
- Providing a new file mode. When message ARI0246D appears, type CHANGE to change the defined medium.

The LRECL and RECFM are automatically set by the log archive utility, and should not be specified on the FILEDEF. You must specify a BLKSIZE in the FILEDEF command. The BLKSIZE must be a multiple of 4 kilobytes, up to a maximum of 28. For optimum performance, follow these guidelines:

- Format the CMS minidisk that contains the log archive with a block size of 4096.
- Use a block size of 28672 (BLKSIZE 28672) for the log archive file.

For more information on the FILEDEF command, see the *VM/ESA: CMS Command Reference* manual for your VM system.

3. Start the application server with LOGMODE=L.
4. Notify the current users that you want to stop the application server (refer to “Log Archiving to Tape” on page 218).
5. Archive the log and stop the application server by typing the operator command:

```
SQLEND LARCHIVE
```

6. You will see the following message:

```
ARI0239I External labeling of this archive is:
          Type:      log archive
          Timestamp: 12-09-92 15:02:23
ARI0252I   Medium:   disk SQLMACH3 12099202 A1
ARI0246D The above information describes the log archive
          about to be done. Enter either:
          CONTINUE   to proceed using the output medium
                    indicated, or
          CHANGE     to change this medium.
```

Notice that “Medium” has changed to disk SQLMACH3 12099202 A1.

7. Type CONTINUE and press ENTER. When the log is completely archived, the following message is displayed:

```
ARI0292I Archive is completed.
```

File Names and File Types Used for Log Archives on Disk: When you direct a log archive to disk, the name of the application server is used as the file name, and the date of the log archive as the file type. The date has the format *mmddyynn*, where the value *nn* is the number of the log archive taken on that day: for example, 01 is the first log archive of the day. (Your system may use a different format.)

You can change the file name and file type of a log archive file when you enter the file mode. If you do, you should choose a new file name and/or file type to help you restore the log archive in the right sequence.

Maintaining Log Archive Disk without Stopping the Application Server: When you direct log archiving to disk, a new file is created for each log archive. If you have a continuous operation environment and the log archive files are not removed

to other storage facilities, the disk eventually gets full. If a “disk full” error occurs, you will not be able to use another user ID to perform maintenance on that disk. You must shut down the application server, and do the maintenance from the machine side. Thus you should periodically transfer the log archive files to other disks to ensure that the disk always has enough space to store another log archive.

You can perform this maintenance through another user ID when the database manager is running, in one of three ways:

1. Use the shared file system.
2. Transfer files from the disk as follows:
 - a. Record the disk address of the disk for log archiving before startup, and wait until you see the messages indicating that the log archive is completed.

- b. Issue the CP DETACH command from the DB2 Server for VM operator console as follows:

```
#CP DETACH daddr
```

where *daddr* is the address of the disk used for log archiving. This disk is now available for maintenance. Link to the disk and transfer the log archive files to other disks.

- c. When maintenance is complete, release and detach the disk used for log archiving.
 - d. Issue the following command from the operator console to link the disk used for log archiving to its original address with write access mode:

```
#CP LINK id vdev1 vdev2 W
```

where *id* is the user ID to which the disk belongs, *vdev1* is the disk address specified in the system directory, and *vdev2* is the original disk address recorded in step 2a.

If the operator forgets to relink the disk and uses the same file mode for the next log archive, a message is displayed to indicate that the device used for log archiving does not exist. The operator should then issue the CP LINK command.

Do not detach and make any changes to the work disk (A-disk), service disk (V-disk), production disk (Q-disk), or other database disks. These disks are crucial for running the database manager. Use a separate disk to store the files created by the log archive.

3. This method does not require that the disk used for log archiving be detached from and relinked to the database machine. The link is kept intact, but the user doing maintenance on the disk between log archives must link to the disk using multiple write mode, because the database manager has a write link to the disk. When you complete the maintenance, release and detach the disk.

In methods 2 and 3 above, the database manager ensures that it is the sole writer of the disk for log archiving at archive time. If this is not the case, it sends a message to the operator, and suspends the archive until the situation is corrected. In method 1, the checking is done by the shared file system.

When a medium for log archiving is decided, the database manager establishes the links, checking that the medium is a disk. If it detects that other machines have the

disk linked in write mode, it displays a list of user IDs with the write link, and the address to which they are linked. The operator is prompted to detach all the write links before proceeding to the log archive, or to quit the link checking and change the medium for the archive. If too many links to the disk exist, including read-only and read/write links, some of the read-only links may have to be detached.

Labeling Your Archive Tapes

Because there are different types of archives, and each may require multiple tape volumes, it is a good idea to label the tapes externally in case you have to restore the database.

When the database manager prompts the operator to mount the tape to record the archive, it also displays a message that includes the date, time, and type of archive (database or log). For example:

```
ARI0239I External labeling of this archive is:
          Type:      log archive
          Timestamp: 12-09-92 14:41:00
ARI0252I Medium:    tape 183
```

The timestamp and type of archive provide identifying information about this archive, and should be written on the external label of each tape reel or cartridge. The label information is provided by the database manager for the first volume of the archive. If your archive requires more than one tape volume, add your own sequential identification to each label (for example, Tape1 of 2, Tape2 of 2).

When the database is restored, the database manager checks if there are any log archives associated with the database archive. If log archives exist, a list of them is displayed, and the time and date of each is provided. The information on the external label can be matched against this list to find the correct tapes to use for the restore.

Recovery Procedures

A system failure is any failure that causes the database manager to end abnormally. Such failures could occur because of an abnormal end of the VM system, or because of error conditions in the database machine.

As long as the current log is available, recovery from system failures is automatic. Even if you are running the database manager in single user mode (SYSMODE=S) with no logging (LOGMODE=N), it can recover any committed updates by using the current log. Restart recovery is performed the next time the application server is started.

If there is a system failure while you are restoring the database, see “Restarting from Failure of a Database Restore” on page 228.

Restarting Procedures

To perform restart recovery procedures, the operator starts the application server with STARTUP set to one of the following values:

W Warm starting with the SQLSTART EXEC

R Restoring from a database manager archive with the SQLSTART EXEC

- U** Restoring from a user archive with the SQLSTART EXEC
- S** Adding dbspaces with the SQLADBSP EXEC
- E** Adding or deleting dbextents with the SQLADBEX EXEC
- I** Reorganizing the catalog indexes with the SQLCIREO EXEC
- M** Doing a catalog migration with the ARISMEX EXEC.

For all these settings, the log is checked at startup to see whether the last run of the database manager left any LUWs in progress. If it did, restart recovery processing starts and the changes made by those LUWs are backed out. Restart recovery processing also ensures that changes made by completed LUWs are, in fact, made.

Restart recovery procedures will not be performed if STARTUP is set to either **C** (for database generation with the SQLDBGEN or SQLDBINS EXECs) or **L** (for log reformatting or reconfiguration, called a COLDLOG operation, with the SQLLOG EXEC).

For both of these settings, the database manager does not check the log, and the LUW recovery processing does not occur.

Restoring the Database

If an unresolvable I/O error occurs on any of the devices that contain the directory or dbextents, the application server ends abnormally. It may be necessary to replace the damaged database minidisk and restore the database from the most recent archive tapes. For more information, see “Replacing a Database Minidisk” on page 232.

If an unresolvable I/O error occurs on a device that contains a log, it is necessary to replace the damaged minidisk. For more information, see “Replacing a Log Minidisk” on page 235.

Selecting the Archive Copy to Use

Locate the last *successful* archive of the database. If the DASD failure occurred while the most recent archive was being taken, then the last successful database archive would be the previous archive copy, not the copy interrupted by the failure.

If you are restoring from the most recent archive and the log minidisk (or at least one of the log minidisks in the case of dual logging) is not damaged, do not perform a COLDLOG before restoring. The current log is required for recovery. In the case of dual logging, where one damaged log has been replaced, restoring the database copies the good log to the new log.

If you are using a back-level database archive and LOGMODE had not been set to L when that archive was taken, or if the minidisks of the log have been changed (regardless of what LOGMODE was set to), you must run a COLDLOG with the SQLLOG EXEC to reformat the logs before restoring. For more information, see “Running the SQLLOG EXEC” on page 241.

You may have to redefine the directory and log disks (or both logs in the case of dual logging) at the same time due to an I/O error. If you are restoring from a user archive, perform a COLDLOG to reformat the logs before continuing with the restore. If you are restoring from an archive produced by database manager

facilities, any attempt to perform a COLDLOG would result in a system error. Instead of performing the COLDLOG after redefining the database minidisks, restore the database. The restore will fail when the database manager tries to read the log. After the restore fails, do a COLDLOG to reformat the logs.

If you are restoring the database by using a database archive and subsequent log archives (LOGMODE=L), locate all the necessary log archives. If you have log archives on disk, your database machine must have access to the CMS minidisks or shared file system directories containing the log archives. If the failure occurred during the archiving of the log, do not use that final log archive tape. The database manager will automatically take another log archive when it is started for the restore.

The steps to be followed to restore your database differ, depending on whether the database had been archived using database manager or user facilities.

Restoring from a Database Manager Archive

Start the application server, with STARTUP=R and LOGMODE=A or L to restore the database using an archive created with database manager facilities. The database manager prompts the operator to mount the database archive tape, and to specify on which unit (cuu) the tape is mounted. Unless a CMS FILEDEF command for ddname ARIARCH with a virtual device address other than TAP1 (181) was submitted before running the SQLSTART EXEC, the archive tape should be mounted on virtual device 181. The tape is then opened, and the database directory and dbextents are copied.

The virtual device address for log archives is 183 (unless a FILEDEF command was issued for ddname ARILARC). Disks that contain log archives must be accessed by the database machine.

Figure 74 shows an example of doing a startup to restore a database that had been archived using database manager facilities.

```
SQLSTART DB(server-name) PARM(STARTUP=R,LOGMODE=L)
```

Figure 74. Starting with STARTUP=R to Restore a Database

Note: In this example, LOGMODE is set to L because the user normally uses log archiving.

The SQLSTART EXEC issues these FILEDEF commands for the database and log archive tapes:

```
FILEDEF ARIARCH TAP1 SL (BLKSIZE 4096 NOCHANGE PERM  
FILEDEF ARILARC TAP3 SL (BLKSIZE 4096 NOCHANGE PERM
```

You can submit your own FILEDEF commands (to override the defaults) before running SQLSTART. You should also submit LABELDEF commands.

Do not specify a VOLID parameter for log archiving. Multiple log archive files can be created on a single run of the database manager. You would want these files to have different VOLIDs.

If you normally run with LOGMODE=A (no log archiving), you would specify LOGMODE=A at startup.

Restoring from a User Archive

Shut down the application server, and restore the database using the same user facilities that created the archive.

Do not restore the database logs. If you accidentally restore the logs, the history area and all the changes to the database recorded in the log, are lost. The database manager uses the history area to track which log archives go with which database archives. The history area must be restored from the backup copy. For more information, see “History Area” on page 244. To recover from accidentally restoring the log, do a COLDLOG to reconfigure the logs before proceeding. That is, run the SQLLOG EXEC without specifying the LOG1 or LOG2 parameters. Respond CONTINUE to message ARI0688D (for single logging) or ARI6129D (for dual logging). Respond 1 to message ARI0944D to reconfigure the log. When the application server is started, the log history area containing the restored database archive can be restored from the version that existed before the COLDLOG. The restored history area will contain a record of all log archives that apply to the restored database, although the log that existed just before the restore will be lost.

After restoring the database directory and dbextents, start the application server with STARTUP=U and LOGMODE=A or L. The operator is asked whether the user restore completed successfully. If the answer is yes, then if LOGMODE=A, the changes in the log are applied to the database; if LOGMODE=L, the database manager takes an archive of the current log, and then restores the log archive tape files that are associated with the user archives. If the operator responds that the user restore was not done, the application server ends, and the operator must take the necessary action to resolve the problem.

When to Use LOGMODE=A

For both database restores (STARTUP=R) and user restores (STARTUP=U), specify LOGMODE=A when you start the application server to have the database manager restore the database without using log archive tape files. When the database is restored, the database manager applies only the changes in the current log to the database. (This is the reason you need to do a COLDLOG if you are not using the most recent database archive, or if you accidentally restored the logs during a user restore: the log does not apply to the older archive.) After completing the restore, the database manager runs with LOGMODE=A.

The database manager still checks whether there are any log archives associated with the database archive. If there are, message ARI0247D is displayed prompting the operator either to keep LOGMODE=A and restore the database without using the log archives, or to switch to LOGMODE=L and use the log archives during the restore. If the decision is made to switch to LOGMODE=L, the database manager runs as if it had been intended to do the restore with LOGMODE=L all along.

When the restore set is complete, the archive that is restored becomes the database archive for the current restore set. A restore set consists of a database archive and the log archives associated with it in the history area -- that is, those log archives that occurred between the database archive and the next restore or COLDLOG or change of log mode.

When to Use LOGMODE=L

Specify LOGMODE=L if you want the database to be restored using log archives. The database manager first restores the database archive and then takes a log archive if information is in the log that was being used when the system failed or was shut down immediately prior to the restore. It then restores the log archives that were taken after the database archive you restored. When the restore is complete, the database manager runs with LOGMODE=L.

Before restoring the database archive and each log archive, the operator is prompted to continue, stop the application server, or end the restore. Usually, the operator responds CONTINUE.

If the operator responds STOP SYSTEM, the application server ends. The next time the application server is warm-started, it will continue restoring the database using the next log archive. If it is restarted to do a restore instead of a warm start, it ignores the first restore, which was stopped, and begins a new one. If it is restarted with STARTUP=C, the application server does the equivalent of an END RESTORE (see below) and then a COLDLOG. (All subsequent log archives are no longer usable.)

The STOP SYSTEM response is used primarily for filtered log recovery. This allows you to stop the application server in the middle of a restore, change the EXTEND input file commands used for filtered log recovery, and continue the restore. For information about filtered log recovery, see the discussion on starting the application server to recover from a DB2 error in the *DB2 Server for VM Diagnosis Guide and Reference* manual.

The END RESTORE response is used primarily for ending a restore before processing a log archive tape that is unusable. A secondary use is to end a restore before processing a log archive that contains a user error.

Attention: If you end a restore, you may lose the ability to use subsequent log archives on a future restore.

For example, suppose you have taken a database archive and six subsequent log archives. If you discover a user error that was recorded in the fourth log archive, restore the database archive and the first three log archives. Enter END RESTORE to avoid processing the fourth, fifth, and sixth log archives. When you end the restore, it may be impossible to restore the database again using the fourth, fifth, and sixth log archives. This would be unfortunate if you had made a mistake and, in fact, should have restored the fourth log archive as well. Thus, before you respond END RESTORE, be sure you have processed the correct number of log archives.

If a situation like the one above occurs, the only way to recover the lost log archives is to restore a back-level database archive. The log archives associated with that database archive must include the ones that were lost. That is, the old database archive must have continuous log archives to the point of the END RESTORE. If it does not, you cannot recover the lost logs. For more information, see “How the Database Manager Uses the History Area” on page 245.

After the restore set is complete, the database archive and log archives that were just restored become the current restore set, unless the restore ended before all log

archives in the restore set were applied. As a final step, the current log is restored if it directly followed the restored log archives.

Restoring Log Archives from Disk

If you have any log archives on disk, the database machine must have access to the minidisks containing these log archives when it begins restoring the database. The database manager issues the FILEDEF commands needed for the log archives on disk, and also checks the timestamp for each log archive file to ensure it matches the timestamp in the history area.

Note: After a restore, the FILEDEF for ARILARC is inoperative. You must specify the FILEDEF again by either:

- Issuing the FILEDEF command.
- Providing a new file mode. To do this, specify CHANGE for message ARI0246D. When this message appears, you can change the defined medium.

Restarting from Failure of a Database Restore

Three types of errors can cause a failure of a database restore operation:

1. System failures, such as power interruptions, or operator or equipment errors that can be corrected. For example, the database manager can end because the wrong tape volume was mounted or a tape drive malfunctioned.

In these error situations, after taking corrective action, you can restart the restore process as follows:

- If you have received message ARI0260I (displayed at the beginning of log recovery), warm-start the application server (STARTUP=W and LOGMODE set to the value used previously). If you are using LOGMODE=L, the database manager continues with the log archive file it was processing when the failure occurred. A warm start saves you processing time for reading and recovering from database and log (if LOGMODE=L) archive files that have already been successfully processed.
 - If you have not received message ARI0260I (or are unsure whether you have received it), restart the restore process specifying the same STARTUP and LOGMODE values you used to initiate the database restore process.
2. A log archive error that can be corrected, or a failure during UNDO/REDO processing.
To deal with a log error that can be bypassed or corrected, refer to the section on recovering from DBSS errors in the *DB2 Server for VM Diagnosis Guide and Reference* manual, especially the discussions on UNDO and REDO processing failures during a restore.
 3. A database or log archive input file error that cannot be corrected, such as a damaged archive tape volume. One of the following situations applies:

- You were using log archiving (LOGMODE=L), and the damaged file is a database archive.

In this situation, you can reset the database to its current state by using a previous database archive and the subsequent log archives (if there are any). You can do this only if the following conditions are met:

- The log archives must be continuous. That is, you have not switched log modes and have not done a COLDLOG (with the SQLLOG EXEC) since the previous database archive.

Note: You can switch from LOGMODE=L to A and then back to L again without breaking the continuity of the log archives, as long as you do not take a database archive while LOGMODE is set to A. For example, suppose you accidentally start the application server with LOGMODE=A instead of L. If you immediately shut down the application server without taking a database archive, the continuity of the log archives is preserved.

- You must not have added dbspaces, added dbextents, or reconfigured the log since the back-level database archive was made. If you have, these changes are not recorded in the log or the log archives, but are recorded in the database directory; thus, if you use the back-level database archive and subsequent log archives to restore the database, the directory will not be synchronized with the database changes, and the restore will fail.

To reset the database using database manager facilities, restart the application server and restore the back-level database using STARTUP=R with LOGMODE set to L. In response to the request to mount the archive tape, mount the tape created by the previous database manager archive. When the database archive tape is restored, the operator is prompted for the subsequent log archives.

To reset the database using user facilities, restore the database using the tape file from the previous user archive. Then start the application server with STARTUP=U and LOGMODE=L. The operator is prompted for the subsequent log archives.

- You were using log archiving (LOGMODE=L), and the damaged file is a log archive.

In this situation, the most current level of the database that you can restore to depends on the last undamaged log archive.

To reset the database, restart the application server with STARTUP=W and LOGMODE set to L. The database manager tries to continue the restore by requesting the log archive that had caused the failure. (The database manager determines where it was interrupted.) Instead of responding CONTINUE, respond END RESTORE to the prompt in message ARI0250D.

- You were not using log archiving. The damaged tape is a database archive tape.

Run the SQLLOG EXEC without specifying the LOG1 and LOG2 parameters to reformat the logs. Respond CONTINUE to message ARI0688D (for single logging) or ARI6129D (for dual logging), then respond 0 to message ARI0944D to reformat the log minidisk (or minidisks). Then restart the restore process using a previous database archive tape.

Note: This previous database archive must have been created by an SQLEND ARCHIVE, SQLEND UARCHIVE, or ARCHIVE command known to have been issued when no application program was accessing the database.

In these situations, all changes made to the database since the database archive was taken are lost. You can reset the database to the consistent state that existed when that database archive tape file was created.

- The log history area was invalid and recovery from the ARIHSDS ARCHIVE failed.

Run the SQLLOG EXEC without specifying the LOG1 and LOG2 parameters to reconfigure the logs. Respond CONTINUE to message ARI0688D (for single logging) or ARI6129D (for dual logging), then respond 1 to message ARI0944D to reconfigure the log minidisk (or minidisks). Then restart the restore process.

In these situations, all changes made to the database since the database archive was taken are lost. You can reset the database to the consistent state that existed when that database archive tape file was created.

Restarting from a System Failure While Archiving

The procedure to recover from a system failure that occurs when the database manager is taking either a log or database archive is essentially the same as any other restart. Because it did not finish, however, the archive that was being written at the time of the failure cannot be used.

Restart the application server with STARTUP=W. If LOGMODE had been set to A or L, specify the same value; if LOGMODE had been set to Y, specify LOGMODE=A.

If the archive in-process had been an automatic archive (started by ARCHPCT), another automatic archive will be initiated immediately when the application server is started again. If it had been started by an ARCHIVE, LARCHIVE, SQLEND ARCHIVE, or SQLEND LARCHIVE command, you must reissue the command when restarting the application server. If it had been an implicit log archive created by issuing SQLEND UARCHIVE with LOGMODE set to L, reissue the SQLEND UARCHIVE command after restarting the application server with LOGMODE=L.

Restarting from Failure of a Database Generation or COLDLOG Operation

If a system failure occurs during database generation or COLDLOG processing, restart the operation (by reissuing the SQLDBINS, SQLDBGEN, or SQLLOG EXEC) after correcting the cause of the failure.

In some cases, storage may need to be reclaimed before continuing processing. For example, an LUW is processing a DROP TABLE statement, a checkpoint is taken during this processing, and a COLDLOG operation immediately follows. If a media failure occurred before the COLDLOG, there is a possibility of rows from the dropped table still existing. However, the entry in the SYSTEM.SYSDROP catalog table no longer exists. To reclaim this storage, the dbspace containing this “dropped” table must be dropped before continuing processing.

Relocating the Database Manager

You can use three ways to move the database manager between system DASD:

- Use DASD Dump Restore (DDR) to move the database manager. With this method, all minidisks must retain the same size and device type. For more information, see “Replacing a Minidisk Using DASD Dump Restore.”
- Use the Data Facility Storage Management Subsystem (DFSMS/VM*) to move the database manager. For more information, see the *DFSMS/VM User's Guide*.
- Archive the database on the original system and restore it on the new system. For more information, see “Replacing a Database Minidisk” on page 232.

Replacing a Minidisk Using DASD Dump Restore

You can use the DASD dump restore (DDR) utility to replace minidisks only if there are no errors on the DASD and you keep the device type, size and virtual address the same as the original minidisk. To change any of these characteristics or recover from a DASD error, follow the procedure in “Replacing a Database Minidisk” on page 232, or “Reconfiguring and Reformatting the Logs” on page 240.

You may want to replace a minidisk either to balance your DASD workload, or to remove all data from a device. To replace it, do the following:

1. Take an archive of the database.
2. Update the MDISK control statements for the affected minidisks in the VM directory entry for the database virtual machine. Alter the original minidisks to different virtual device addresses, and add the new minidisks using the original virtual device addresses. Write down the original and new addresses as you will need them later. The new minidisks must be the same size and device type as the original ones.
3. Log on to the database virtual machine, and IPL (initial program load) CMS.
4. Do the following for each replaced minidisk:
 - a. Issue the following CP command to make the original minidisk read-accessible to the virtual machine:

```
LINK * cuu1 cuu2 R
```

where:

- the * assumes that the minidisk is defined in the VM directory for this virtual machine.
 - the *cuu1* is the virtual device address as specified with the MDISK statement.
 - the *cuu2* is usually the same as *cuu1*.
 - the R indicates that read mode access is required for this minidisk.
- b. Issue the following CP command to give the virtual machine write access to the new minidisk:

```
LINK * cuu3 cuu4 W
```

where:

- the * assumes that the minidisk is defined in the VM directory for this virtual machine.

- the *cuu3* is the virtual device address as specified with the MDISK statement.
 - the *cuu4* is usually the same as *cuu1*.
 - the W indicates that write mode access is required for this minidisk.
- c. Issue the following command to copy the minidisk using the DDR utility:

```
DDR
```

The DDR utility prompts you for your instruction statements. Enter the information that you wrote down from the MDISK statements for the input and output devices, CONS for the printer and copy the original minidisk to the new one, as shown below.

```
SYSPRINT CONS
INPUT cuu2 type
OUTPUT cuu4 type
COPY ALL
```

where:

- the *cuu2* and *cuu4* are the second virtual device addresses specified in the link commands issued earlier.
- the *type* is the device type as specified with the MDISK statement.

Respond YES to the prompts about the labels on the minidisks.

Attention: Be sure that you are accessing the correct minidisks before you respond YES to the label prompts.

- d. Issue the following command to detach the original minidisk:

```
DETACH cuu2
```

where *cuu2* is the virtual device address as specified on the LINK command and INPUT statement.

5. Update the MDISK statements for the affected minidisks in the VM directory entry for the database machine. Remove the MDISK statements for the original minidisks.

Replacing a Database Minidisk

This section describes how to replace a database directory (ddname BDISK) minidisk or dbextent (ddname DDSK*n*) minidisk. You can replace directory or dbextent minidisks only if you have been archiving your database.

You may want to replace a database minidisk because:

- The minidisk is damaged because of an unrecoverable DASD error.

You may need to replace the directory or database minidisks, because one or both were damaged. In this situation, if you are running with dual logging and only one of the logs is damaged, replace the directory or database minidisks **first** by following the steps below. Then replace the log minidisk by following the procedures in “Log Reconfiguration” on page 240. Finally, restore the database by following the procedures in “Replacing a Log Minidisk” on page 235.

- You want to move your minidisks to a different device type.

If you are replacing all the database minidisks (as you might when moving the database to a different device type), replace the log minidisks **first**. Follow the procedures in “Log Reconfiguration” on page 240.

- You want to balance your DASD workload.

Use the instructions below if you are moving the directory or database minidisks. If you are moving your logs, refer to “Log Reconfiguration” on page 240.

To replace directory or database minidisks, do the following:

1. Create a database manager archive of the database. For more information, see “Archiving Procedures” on page 211. The archive is required for the steps below.
2. Write down the name of the application server and the *ddnames* of the database minidisks being replaced. These will be required in the steps below.
3. Update the MDISK control statements for the affected database minidisks in the VM directory entry for the database virtual machine. Figure 90 on page 297 describes the MDISK control statements for database minidisks. You must not change the virtual device addresses for the MDISK statements that are updated.

Note the virtual addresses of the minidisks being replaced (for use below). You must ensure that the minidisks are either same size or larger than the old minidisks. If you are replacing the minidisk with one on the same device type, define the same number of cylinders or blocks.

If you are replacing the minidisk with one on a different device type, you should define the new minidisk to be slightly larger than the old. Because of rounding that occurs in the space allocation algorithms, it is nearly impossible to define minidisks on two different device types so that the database manager considers them to be equal. If you define the new minidisk so that it is close to the same size as the old one, the restore can fail because of lack of space. When you do the restore (in a later step), the database manager uses the space it needs and ignores the rest. It will not use the blocks on the new minidisk that exceed the number of blocks on the old minidisk. For help in estimating the equivalent sizes of minidisks on various devices, refer to “Determining Equivalent Minidisk Sizes on Different Device Types” on page 452.

4. Log on to the database virtual machine and do an IPL of CMS.
5. Use the CMS QUERY DISK command to obtain a disk access letter not currently in use. (If you have no disk accessed as C, note this for use below).
6. Do the following for each replaced database minidisk:
 - a. Issue the following CP command to give the virtual machine write access to the minidisk:

```
LINK * cuu1 cuu2 W
```

- The * assumes that the minidisk is defined in the VM directory entry for this virtual machine.
- The *cuu1* and *cuu2* should both be the virtual device address as specified with the MDISK statement (in step 3); that is *cuu1=cuu2*.
- The W indicates that write mode access is required for this minidisk.

b. Issue the following CMS command to initialize and label the minidisk:

```
FORMAT cuu1 access-letter (BLKSIZE size)
```

- The *cuu1* is the virtual device address you used on the CP LINK command above.
- The *access-letter* is the disk access letter you obtained in step 5 on page 233.
- The size is 512 if the minidisk is the database directory file. Otherwise, specify 4096.
- The FORMAT command prompts you whether to erase all files on the minidisk. Reply YES.

Attention: Be sure that you are accessing the correct minidisk before you respond YES to the FORMAT prompt.

- The FORMAT command prompts you for the minidisk volume label (disk label). Enter any valid volume label value. Because FILEDEF ddnames are limited to 8 characters, and minidisk volume labels are limited to 6 characters, the database manager uses the following conventions for the minidisk labels:
 - BDISK for the database directory minidisk
 - DDK*n* for dbextent minidisks (*n* is the *n*th dbextent).

c. Issue the following CMS commands to allocate the minidisk as a block I/O file:

```
RESERVE filename filetype filemode
```

- For *filename*, use the RESID.
- For *filetype*, use the FILEDEF ddname of the file, where the ddname is as follows:
 - BDISK for the database directory minidisk
 - DDSK*n* for dbextent minidisks (*n* is the *n*th dbextent).
- For *filemode*, use the access-letter that you entered above for the CMS FORMAT command.
- The RESERVE command prompts you whether to erase all files on the minidisk. Reply YES. Be sure that you have specified the file mode letter that matches the CMS FORMAT command.

d. Issue the following CMS command to release the minidisk:

```
RELEASE cuu1
```

The *cuu1* is the virtual device address that you previously specified on the LINK and FORMAT commands.

7. Restore the database archive taken in substep 1 on page 233. For more information, see “Restoring the Database” on page 224.

Replacing a Log Minidisk

If you are relocating the log disks to another device because of disk migration or to control device utilization, and the target log disk is the identical device type and size as the source log disk and the source log disk is not damaged, you can use the SQLCDBEX EXEC to move the log disk. See “Moving Log Disks” on page 179 for more information.

This section describes how to replace a log minidisk (ddname LOG1 or LOG2). You would replace a log minidisk if:

1. The minidisk is damaged by an unrecoverable DASD error.
2. You want to change the size of your logs.
3. You want to move your minidisks to a different device type.

To replace log minidisks:

1. If you are replacing the only log (for single logging) or both logs (for dual logging), take a database archive if you are running with LOGMODE=A or L, because the contents of the log, including the history area, will be lost. If you are dual logging and you are only replacing one log, the archive is not lost.
2. Update the MDISK control statements for the log minidisks in the VM directory entry for the database machine. For a description of these statements, see Figure 90 on page 297.
3. If you are replacing the only log (for single logging) or both logs (for dual logging), follow the procedures on “Log Reconfiguration” on page 240.
4. If dual logging and you are only replacing one log, do the following for the replaced log:

- a. Use the CMS QUERY DISK command to obtain a disk access letter not currently in use.
- b. Issue the following CP command to give the virtual machine write access to the minidisk:

```
LINK * cuu1 cuu2 W
```

- The * assumes that the minidisk is defined in the VM directory entry for this virtual machine.
 - The *cuu1* and *cuu2* should both be the virtual device address as specified with the MDISK statement (in step 3 on page 233); that is *cuu1=cuu2*.
 - The W indicates that write mode access is required for this minidisk.
- c. Issue the following CMS command to initialize and label the minidisk

```
FORMAT cuu1 access-letter (BLKSIZE 4096
```

- The *cuu1* is the virtual device address you used on the CP link command above.
- The *access-letter* is the disk access letter you obtained in 4a.
- The FORMAT command prompts you whether to erase all files on the minidisk. Reply YES.

Attention: Be sure that you are accessing the correct minidisk before you respond YES to the FORMAT prompt.

- The FORMAT command prompts you for the minidisk volume label (disk label). Use LDISK1 for the first log minidisk, or LDISK2 for the second log minidisk.
- d. Issue the following CMS command to allocate the minidisk as a block I/O file:

```
RESERVE filename filetype filemode
```

- For *filename*, use the RESID.
 - For *filetype*, use LOGDSK1 for the first log minidisk or LOGDSK2 for the second log minidisk.
 - For *filemode*, use the access-letter that you entered above for the CMS FORMAT command.
 - The RESERVE command prompts you whether to erase all files on the minidisk. Reply YES. Be sure that you have specified the file mode letter that matches the CMS FORMAT command.
- e. Issue the following CMS command to release the minidisk:

```
RELEASE cuu1
```

The *cuu1* is the virtual device address that you previously specified on the LINK and FORMAT commands.

Recovering to a Secondary System

To be able to recover in cases where the original database minidisks are not available (for example, in an offsite disaster recovery situation), you should make a copy of the ARIHSDS ARCHIVE file from the application server's A-disk after every log archive or database archive. You would then recover to a secondary system. The secondary system must have the same dbextent configuration and number of logs as the original system.

If you have been running with LOGMODE=A and need to recover to a secondary system, do a COLDLOG RECONFIGURE to initialize the log (see "Log Reconfiguration" on page 240). Then restore the most recent archive on the secondary system.

If you have been running with LOGMODE=L and need to recover to a secondary system:

1. Do a COLDLOG RECONFIGURE to initialize the log (see "Log Reconfiguration" on page 240).
2. Replace both the ARIHSDS ARCHIVE and ARIHSDS PRECLDLG files on the secondary server's A-disk with the ARIHSDS ARCHIVE file you copied after the latest database or log archive of the original system.
3. Restore the most recent archive on the secondary system.

Chapter 10. Special Topics in Recovery Design

This chapter describes how to switch log modes, how to use dual logging, how to reconfigure and reformat the logs, and how to use nonrecoverable storage pools.

Switching Log Modes

In general, you should not switch indiscriminately between log modes Y, N, L, and A: pick one mode and stick to it. However, switching to another mode may at times be required. (See “Choosing a Log Mode” on page 208 for description of log modes.)

From LOGMODE=A

To switch to LOGMODE=Y or N:

1. Issue either an `SQLEND ARCHIVE` or an `SQLEND UARCHIVE` command. With `SQLEND ARCHIVE`, a database archive is automatically taken, then the application server shuts down; with `SQLEND UARCHIVE`, the application server shuts down immediately, then you take the user archive (using your own facilities).
2. Start the application server with `STARTUP=L` and `LOGMODE=Y` to perform a `COLDLOG` to reformat the log.
3. Run the `SQLLOG EXEC`, omitting the `LOG1` and `LOG2` parameters, to reformat the logs. Respond `CONTINUE` to message `ARI0688D` (for single logging) or `ARI6129D` (for dual logging). Respond `0` to message `ARI0944D` to reformat the log minidisks. For more information, see “Running the `SQLLOG EXEC`” on page 241.
4. Start the application server with `STARTUP=W` and `LOGMODE=Y` or `N`.

To switch to LOGMODE=L:

1. Issue either an `SQLEND ARCHIVE` or an `SQLEND UARCHIVE` command. With `SQLEND ARCHIVE`, a database archive is automatically taken, then the application server shuts down; with `SQLEND UARCHIVE`, the application server shuts down immediately, then you take the user archive. In either case, this database archive serves as the starting point for subsequent log archives.

You do not have to take this database archive under either of the following two conditions:

- You have already taken one, and have been running with `LOGMODE=A` since that archive.
- You have done a restore that finished without interruption, and have done nothing to break the continuity of the restore set. (For information on how the continuity of the restore set can be broken, see “History Area” on page 244.)

In either of these situations, the database archive you took (or restored) is in the current restore set.

2. Start the application server with `STARTUP=W` and `LOGMODE=L`.

From LOGMODE=L

To switch to LOGMODE=Y or N:

1. Shut down the application server by issuing an SLEND LARCHIVE operator command to save the log.
2. Run the SQLLOG EXEC, omitting the LOG1 and LOG2 parameters, to reformat the logs. Respond CONTINUE to message ARI0688D (for single logging) or ARI6129D (for dual logging). Respond 0 to message ARI0944D to reformat the log minidisks. For more information, see "Running the SQLLOG EXEC" on page 241.
3. Start the application server with STARTUP=W and LOGMODE=Y or N.

To switch to LOGMODE=A:

1. Shut down the application server by issuing an SLEND LARCHIVE operator command to save the log.
2. Start the application server with STARTUP=W and LOGMODE=A.

You will be warned that the continuity of the log archives will be broken.

Switching the log mode when you have been using log archiving will interrupt the continuity of the log archives, unless all you do is switch from LOGMODE=L to A and then back again without taking a database archive. (This protects you from losing a sequence of log archives if you accidentally set LOGMODE to A.) If the continuity is broken and work is done on the database, you will not be able to restore the database to its current level by using database and log archives taken prior to the break. Figure 75 shows this situation:

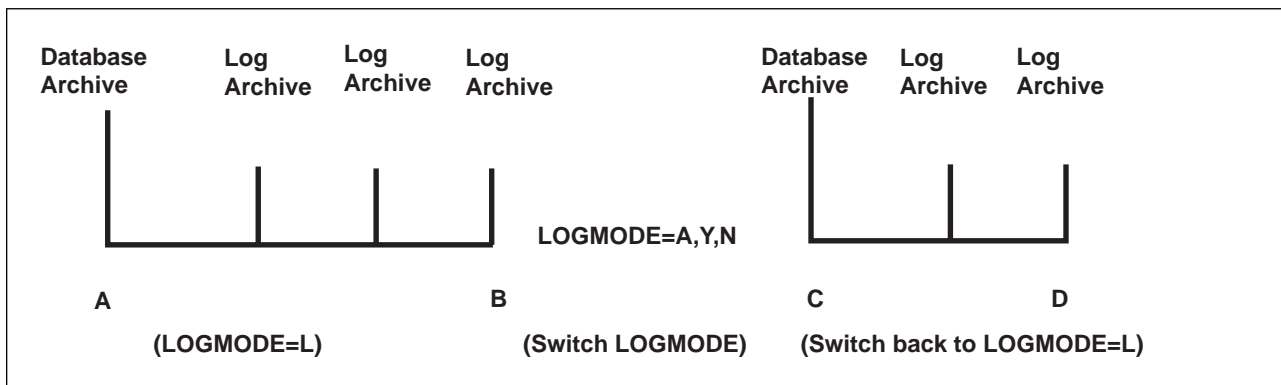


Figure 75. Log Archive Continuity

In the above diagram:

- D is the current database status.
- If you use the database archive taken at A and subsequent log archives, you can restore the database only to point B. All changes between points B and D are lost.
- If you use the database archive taken at C and subsequent log archives, you can restore the database to point D.

From LOGMODE=Y or N

To switch to LOGMODE=A:

1. Start the application server with STARTUP=W, LOGMODE=Y, and SYSMODE=M.
2. Issue either an SLENDD ARCHIVE or an SLENDD UARCHIVE command. With SLENDD ARCHIVE, a database archive is automatically taken, then the application server shuts down; with SLENDD UARCHIVE, the application server shuts down immediately, then you take the user archive (using your own facilities).
3. Start the application server with STARTUP=W and LOGMODE=A.

To switch to LOGMODE=L:

1. Start the application server with STARTUP=W, LOGMODE=Y, and SYSMODE=M.
2. Issue either an SLENDD ARCHIVE or an SLENDD UARCHIVE command. With SLENDD ARCHIVE, a database archive is automatically taken, then the application server shuts down; with SLENDD UARCHIVE, the application server shuts down immediately, then you take the user archive.

The continuity of the log archives will have been interrupted by any work that was done while LOGMODE was set to Y or N, so you must take a new database archive. This database archive will serve as the starting point for subsequent log archives.

3. Start the application server with STARTUP=W and LOGMODE=L.

Using Dual Logging

The dual logging option (initialization parameter DUALLOG=Y) protects the database in case of a DASD failure on the log disk. With single logging, any I/O error on the log minidisk causes the database manager to end. With dual logging, database updates are recorded in two log minidisks. This reduces the risk of losing the log, as an unrecoverable error is unlikely to occur on both log minidisks at the same time. The database manager continues running as long as it can read and write from one of the log minidisks.

Chapter 2, Planning for Database Generation describes how to generate a database with either single or dual logging. It also covers various log considerations such as log size and log placement.

To switch between single and dual logging, use the SQLLOG EXEC. For more information, see "Reconfiguring and Reformatting the Logs" on page 240 .

Using the VM DUPLEX Command

Database availability and performance can be improved by making use of the VM Dual Copy function. This function allows two devices attached to the same 3990 Model 3 device to operate in duplex mode. This means that all data written to the primary device is also written to the secondary device. The VM DUPLEX command performs the duplex control operations.

Duplexing the log provides the same benefits as dual logging, but with better performance because it is handled by the operating system. You can also duplex the entire database to provide additional benefits. However, to be effective, *all* directory disks and dbextents must be duplexed.

Note: If the DUPLEX command is used, the PACE parameter is important to ensure continuous database activity during recovery. The PACE parameter indicates the rate at which synchronization of the two volumes is to take place.

For more information on the Dual Copy function, see the *IBM 3990 Storage Control Planning, Installation, and Storage Administration Guide* manual. For more information on the DUPLEX command, see the *VM/ESA: CP Command and Utility Reference* manual.

Reconfiguring and Reformatting the Logs

During the life of a database, you may occasionally need to change the physical configuration of the logs. You can use SQLCDBEX to move the logs. See “Moving Log Disks” on page 179 for more information. At other times, you will need to reset the contents of the log logically. This is referred to as *log reformatting* and is required, for example, when you switch from LOGMODE=A or L to LOGMODE=Y or N.

log reconfiguration and reformatting

In this section, the term **log reconfiguration** means that the history area has been erased. **Log reformatting** means that history area has not been erased. Both erase the current database updates saved in the log.

The operation that performs both log reconfiguration and log reformatting is called a COLDLOG. To do a COLDLOG, run the IBM-supplied SQLLOG EXEC. For more information, see “Running the SQLLOG EXEC” on page 241.

Log Reconfiguration

You must reconfigure the logs if you do any of the following.

- Change the size of your logs or switch to a different device type
- Change the location of one log (for single logging) or both logs (for dual logging). You may be able to use the SQLCDBEX exec to move the log, and if so, a COLDLOG is not necessary. See “Moving Log Disks” on page 179 for more information.
- Switch from single logging to dual logging. If your source log disk is undamaged, and the target disk is the identical type and size, SQLCDBEX can be used to copy the log and preserve its contents.

To reconfigure the logs:

1. Take a database archive if you are running with LOGMODE=A or L, because the contents of the log (including the history area) will be erased.
2. Update the MDISK control statements for the log minidisks in the VM directory entry for the database virtual machine. For a description of these statements, see Figure 90 on page 297.

3. Start the SQLLOG EXEC. For information, see “Running the SQLLOG EXEC” on page 241.

If you are using archiving (LOGMODE=A or L), you must create a new database archive immediately after running the SQLLOG EXEC. This ensures that the archive copy of the database correctly reflects the size of the logs and whether or not dual logging is in effect. Since the SQLLOG EXEC changes the LOGMODE to Y, you cannot set LOGMODE to L or A until you have taken a database archive. To create the new archive:

1. Start the application server in multiple user mode, and specify LOGMODE=Y, STARTUP=W, and SYSMODE=M (both are defaults) on the SQLSTART EXEC.
2. When startup is complete, enter the SLENDD ARCHIVE or SLENDD UARCHIVE command. If you issue SLENDD UARCHIVE, take a user archive when the application server is shut down.
3. Start the application server with your normal LOGMODE.

Log Reformatting

You must reformat the logs if you do any of the following:

- When you switch from LOGMODE=A or L to Y or N.
- When you cannot do a warm start because of a logical error in the current log
- When you want to avoid log recovery in restoring a database from a back-level database archive.
- When you switch from dual logging to single logging.

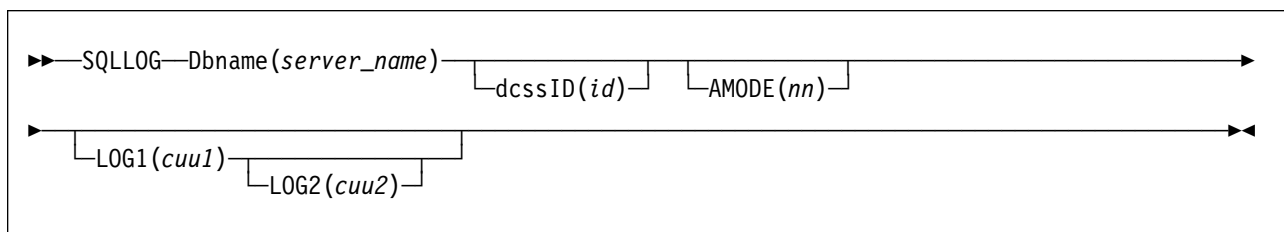
To reformat the logs:

1. Take an archive if you are running with LOGMODE=A or L, because the contents of the log will be erased (but not the history area). If you are switching from LOGMODE=L to Y or N, you can take either a log archive or a database archive. If you are switching from dual logging to single logging and you use LOGMODE=L, you can take a log archive. For other log reformatting situations, take a database archive.
2. Start the SQLLOG EXEC. For information, see “Running the SQLLOG EXEC.”

Running the SQLLOG EXEC

The SQLLOG EXEC begins the COLDLOG operation by starting the application server (with the SQLSTART EXEC) in single user mode (SYSMODE=S) with STARTUP=L, LOGMODE=Y, and DUALLOG=Y (if two logs are defined) or DUALLOG=N (if one log is defined).

This EXEC resides on the service minidisk (V-disk); it can be run only from a database machine. Its format is:



Dbname(*server_name*)

The *server_name* variable identifies the application server.

dcssid(*id*)

Specify this parameter only if you have created discontinuous saved segments for the DB2 Server for VM code and want to use them; otherwise omit it. You can specify ID instead of DCSSID; no other abbreviation is valid. For information about starting the application server to use a saved segment, see Chapter 8, "Saved Segments" on page 181.

AMODE(*nn*)

This parameter is optional. It specifies the type of addressing the database manager runs in.

The AMODE(24) option only needs to be specified when user-written exits do not support 31-bit addressing. In this case, AMODE(24) must be specified when running the SQLLOG EXEC. For a description of this parameter, see "AMODE" on page 60.

LOG1(*cuu1*)

This parameter must be specified if you are reconfiguring the logs and not just reformatting them. *cuu1* is the virtual device address of your first log minidisk (ddname LOGDSK1).

LOG2(*cuu2*)

This parameter must be specified if you are reconfiguring the logs (not just reformatting them) and want dual logging. Omit it if you want only single logging. *cuu2* is the virtual device address of your second log minidisk (ddname LOGDSK2).

Notes:

1. Specifying LOG2 and omitting LOG1 is an error condition.
2. Specifying LOG1 and omitting LOG2 causes the deletion of any entry in the database *resid* SQLFDEF CMS file for a second log minidisk (LOGDSK2), and causes the application server to be started with DUALLOG=N (single logging).
3. If both LOG1 and LOG2 are omitted, the log minidisk entries in the database *resid* SQLFDEF CMS file determine whether the COLDLOG operation is started with DUALLOG=N or Y.
4. The LOG1 *cuu1* value replaces the virtual device address of the first log minidisk entries in the *resid* SQLFDEF CMS file.
5. The LOG2 *cuu2* value replaces the virtual device address of the second log minidisk entries in the *resid* SQLFDEF CMS file if entries are present, or adds the appropriate entries if none yet exist.
6. Message ARI2010I is issued if an abnormal end occurred the last time the application server was stopped. In this case, the log or logs are required for a warm start of the application server.
7. The SQLLOG EXEC prompts you to either reformat or reconfigure the log minidisks. Reformatting the log erases only the log data. The history data is saved. Reconfiguring the log erases both the log data and the history data. (For more information, see "History Area" on page 244.)

Respond CONTINUE to message ARI0688D (for single logging) or ARI6129D (for dual logging). Respond 0 to message ARI0944D to reformat the log, or 1 to reconfigure the log.

Respond 1 to message ARI0944D only if:

- You are moving the logs to a new minidisk. Do this even if you are using dual logging but are only moving one log.
- You are changing the size of the log. (If you use dual logging, you must increase the size of both logs, as their sizes must always be identical.)
- You are adding a log (switching from single logging to dual logging). In this situation, respond 1 (YES) to reconfigure both the log that you are adding and the one that already exists.

Attention: Be sure that you are accessing the correct minidisk before you respond 1 to message ARI0944D to reconfigure the log minidisk.

8. If the SQLLOG EXEC (or the database manager itself, during the COLDLOG) ends with an error, rerun it after correcting the error condition.

For example, suppose you are using database archiving (LOGMODE=A) and dual logging. Your userid is USER1. Another user of the database, USER2, has committed a LUW that made many erroneous updates to the database, and you want to restore the database from a back-level database archive copy. To avoid the log recovery processing that this would entail, you decide to reformat the logs:

1. Take a database archive in case anything goes wrong:

```
SQLEND ARCHIVE DVERIFY
```

2. Reformat the logs of the database (USER1DB):

```
SQLLOG DB(USER1DB)
```

3. Respond 0 to message ARI0944D to reformat the log minidisks.

Now suppose that USER2 makes many changes to the database and consistently causes the database manager to take online archives because the log has been filled. You decide to increase the size of the logs:

1. Issue an SQLEND ARCHIVE DVERIFY command, knowing that the history area is going to be erased.
2. Update the MDISK directory entries for the log minidisks, which are defined at virtual device addresses 502 and 503.

3. Start SQLLOG to reconfigure the logs:

```
SQLLOG DB(USER1DB) LOG1(502) LOG2(503)
```

4. Respond 1 to message ARI0944D to reconfigure the log minidisks because the physical size of the logs has been altered. Reconfiguring the logs allows the database manager to use the additional space.

Now suppose that USER2 has added so much data to the database that you must add more dbextents to the database. Unfortunately, your computer center does not allow use of any more DASD space. You therefore decide to switch to single logging temporarily and use the freed DASD space for another dbextent. To reduce the risk of DASD failure on the remaining log, you switch to log archiving and take frequent log archives:

1. Take a database archive because you are switching to single logging mode:

SQLEND ARCHIVE DVERIFY

2. Delete the MDISK directory entry for the 503 minidisk, but do not change the entry for the 502 minidisk.
3. Issue the SQLLOG EXEC to change to single logging:

```
SQLLOG DB(USER1DB) LOG1(502)
```
4. Respond 0 to message ARI0944D to reformat the log minidisks, so that the history area is preserved. Even though this is a log reconfiguration, there is no need to FORMAT and RESERVE the remaining log minidisk, as you have not increased the size of the remaining log or moved it.

Switching Log Data between Logs

If you are recovering using dual logging, and the primary log does not contain readable data, the database manager automatically switches to the secondary log. If the primary log can be read but it is corrupt, follow these steps to recover using the data on the secondary log:

1. Replace the primary log disk by following the procedures in “Replacing a Log Minidisk” on page 235. This uses the CMS FORMAT and RESERVE commands on the minidisk containing the primary log.
2. Warm start the application server.

This will cause it to compare the two logs, and copy the contents of the secondary log onto the primary log minidisk.

If this procedure fails to recover the database, you must restore it from the last database archive, or do a COLDLOG operation to reformat the logs.

History Area

The distinction between a log reconfiguration and log reformatting is the effect each has on an internally used portion of the log known as the history area. This is a portion of the log that the database manager uses to keep track of recovery events such as database archives, log archives, restores, COLDLOGs, and the switching of log modes. Log reconfiguration causes the history area to be erased; log reformatting does not.

If the history area is erased, the database manager cannot tell which log archives belong with which database archives, or if the continuity of log archiving was broken. In fact, it cannot tell whether you were using log archiving at all, so the database manager cannot allow you to restore the database using a database archive and subsequent log archives.

As the final step in a database, user, or log archive, the database manager archives the newly updated history area into the archive file ARIHSDS ARCHIVE on the database machine work disk. This copy will be used to restore the history area in the event that it is corrupted. This reduces the risk of not being able to do a restore using log archives because of an unrecoverable error in the history area.

Before doing a log reconfiguration, the database manager saves the history area in file ARIHSDS PRECLDLG on its work disk. If a restore is done immediately after a log reconfiguration (meaning that the new history area is empty), this old history area will automatically be restored. This area may also be used for restoring back-level databases: if the database to be restored is found here, not in the

current history area, you may choose to restore the old area to the current log. If you do so, associated log archives may be applied because they will now be found in the log's history area. Note that, because the old history area is restored to the log, no records of any archives taken after the COLDLOG reconfiguration will be available.

You can always restore the database from a back-level or a current database archive. But if the history area is erased, you lose the ability to restore the database using any log archive taken prior to this erasure. Also, if the database archive was taken online (with the ARCHIVE command), the database could be restored to an inconsistent state. For example, a LUW could have made changes before the archive was taken, and then be rolled back after the archive finished. When the database archive is restored, the changes made before the archive was taken are in the database, but any changes made after the archive will be lost.

Whenever a log minidisk is reconfigured, by you directly or by the SQLLOG EXEC, the history area is lost. (The SQLLOG EXEC always prompts you with message ARI0944D before erasing the history area.)

How the Database Manager Uses the History Area

The following description is not intended to be comprehensive; it only provides general background information about log archive recovery processes using the history area.

To display the history area, issue the operator command SHOW LOGHIST. For a discussion of this command, see the *DB2 Server for VSE & VM Operation* manual.

Suppose that you take a database archive (using either database manager or user facilities), followed by four log archives. The history area of the log would contain one record for each of these events:

```
Database Archive 1
Log Archive 1
Log Archive 2
Log Archive 3
Log Archive 4
```

The records in the history area itself would be in an internal (unreadable) format. For ease of description, the records of the history area are shown in an externalized form.

If you now request another database archive, then because the database manager is running with LOGMODE=L, it first takes another log archive of the current log (Log Archive 5 in the example below). If you then take three subsequent log archives the history area would contain the following records:

```
Database Archive 1
Log Archive 1
Log Archive 2
Log Archive 3
Log Archive 4
Log Archive 5
Database Archive 2
Log Archive 6
Log Archive 7
Log Archive 8
```

When you take an archive, the database manager generates identification information based on the processor's time-of-day clock. When you restore the database, the database manager reads this information in the database archive tape file before it looks at the history area.

During a restore, you may be requested to take a log archive of the current log to save the changes up to the point of the restore. When you restore the database from the restore set containing this log archive (and actually restore the log archive), it is erased from the log history's restore set because it is put back into the current log.

When the database manager identifies the database archive tape that is being restored, it writes a record in the history area to indicate that a restore is being done. Next it looks for the corresponding database archive record in the history area.

For example, suppose you start the application server with STARTUP=R, and mount the Database Archive 2 tape file. The database manager looks for the corresponding record in the history area (by searching in reverse chronological order, from the most recent to the least recent entries). When it finds it, it determines the log archives associated with the database archive by reading forward in the history area until the RESTORE record is reached. Log Archive 9 is taken before the restore set is determined. This set of records is referred to as the restore set.

	Read back to the Database Archive Record:	Read forward to identify associated log records:
Write a RESTORE record:		
Database Archive 1	Database Archive 1	Database Archive 1
Log Archive 1	Log Archive 1	Log Archive 1
Log Archive 2	Log Archive 2	Log Archive 2
Log Archive 3	Log Archive 3	Log Archive 3
Log Archive 4	Log Archive 4	Log Archive 4
Log Archive 5	Log Archive 5	Log Archive 5
Database Archive 2	Database Archive 2 <---	Database Archive 2 <---
Log Archive 6	Log Archive 6	Log Archive 6 <---
Log Archive 7	Log Archive 7	Log Archive 7 <---
Log Archive 8	Log Archive 8	Log Archive 8 <---
Log Archive 9	Log Archive 9	Log Archive 9 <---
RESTORE	RESTORE	RESTORE

The database manager copies the restore set records after the RESTORE record:


```

Database Archive 1
Log Archive 1
Log Archive 2
Log Archive 3
Log Archive 4
Log Archive 5
Database Archive 2 <---
Log Archive 6 <--- Restore set
Log Archive 7 <---
Log Archive 8 <---
Log Archive 9 <---
RESTORE
Database Archive 2 <---
Log Archive 6 <--- Restore set copied forward
Log Archive 7 <---
Log Archive 8 <---
Log Archive 9 <---

```

The database manager then displays the restore set to the console using messages. If you restore all the log archives associated with the database archive, the history area remains as shown above, except that Log Archive 9 is erased from the restore set copied forward when it is restored to the current log. If you respond END RESTORE to one of the prompts, the database manager deletes the remaining log archive records from the history area. For example, suppose you respond END RESTORE after only two of the log archives are processed. The final two log archives in the history area are deleted:

```

Database Archive 1
Log Archive 1
Log Archive 2
Log Archive 3
Log Archive 4
Log Archive 5
Database Archive 2
Log Archive 6
Log Archive 7
Log Archive 8
Log Archive 9
RESTORE
Database Archive 2
Log Archive 6 <--- Only two log archives are restored
Log Archive 7 <---

```

After the restore is ended, processing continues and two more log archives are taken. Now the history area looks like this:

```

Database Archive 1
Log Archive 1
Log Archive 2
Log Archive 3
Log Archive 4
Log Archive 5
Database Archive 2
Log Archive 6
Log Archive 7
Log Archive 8
Log Archive 9
RESTORE
Database Archive 2
Log Archive 6
Log Archive 7
Log Archive 10    <--- New log archives
Log Archive 11    <---

```

If you must again restore the database and use Database Archive 2, the restore set will contain Log Archives 6, 7, 10, and 11. Because the database manager determines the restore set by scanning backwards in the history area until it finds a corresponding database archive record, the original Database Archive 2 record (the one before the RESTORE) is never reached. Consequently, it is impossible to use Log Archive 8 or 9 when restoring the database from Database Archive 2.

The only way to restore Log Archive 8 or 9 after you responded END RESTORE is to restore from a back-level database archive. This archive must have continuous log archives to the log archive you want to restore.

In our example, to restore the database to its status immediately before the restore, start the application server to do a restore, and restore Database Archive 1. The database manager scans backwards to the first occurrence of a Database Archive 1 record. (There is only one occurrence.) When it finds the record, it scans forward in the history area until it either reaches the end of the history area or until it finds:

- A record that indicates a COLDLOG was taken.
- A record that indicates LOGMODE was switched to N.
- A record that indicates LOGMODE was switched to Y.
- A RESTORE record.
- Two database archive records in a row (no log archive records in between).
- Records that indicate a switch to LOGMODE=A, and a database archive while using LOGMODE=A. (When the database is archived, the log is reclaimed without a log archive. This breaks the continuity of the log archives.)

These records indicate a break in the continuity of the log archives. If you restore Database Archive 1 in our example, the restore set copied forward in the history area includes Log Archive 9:

```

Database Archive 1 <---
Log Archive 1 <---
Log Archive 2 <---
Log Archive 3 <--- New
Log Archive 4 <--- Restore
Log Archive 5 <--- Set
Database Archive 2 <---
Log Archive 6 <---
Log Archive 7 <---
Log Archive 8 <---
Log Archive 9 <---
RESTORE <--- Indicates end of restore set
Database Archive 2
Log Archive 6
Log Archive 7
Log Archive 10
Log Archive 11
RESTORE
Database Archive 1 <---
Log Archive 1 <---
Log Archive 2 <---
Log Archive 3 <---
Log Archive 4 <--- Restore Set Copied Forward
Log Archive 5 <---
Log Archive 6 <---
Log Archive 7 <---
Log Archive 8 <---
Log Archive 9 <---

```

During the actual restore, only the log archives are applied. Database Archive 2 is skipped because all the change activity is recorded in the continuous log archives.

When the database is restored, it reverts to the state it was in before the first restore. The changes recorded in Log Archives 10 and 11 are lost.

The important points to remember from this discussion are:

- You can issue the `SHOW LOGHIST` command to determine what log archives will be restored. To determine the restore set, scan backwards in the command output until the appropriate database archive is reached; then scan forward to determine what log archives are associated with that database archive. When you reach a recovery event that breaks the continuity of the log archives, you have reached the end of the restore set.
- If you responded `END RESTORE` and later want to restore the subsequent log archives, you must restore a back-level database archive whose associated log archives include those that were skipped by the issuing of the `END RESTORE` command.

If the database manager cannot find the database archive in the current history area or in the history area saved by the last `COLDLOG` reconfiguration (if one exists), a message is displayed saying the database archive is unknown. You are given the opportunity to do a `COLDLOG` (if one has not yet been done) to reformat the log. The `COLDLOG` is necessary because, since the database manager cannot determine a recovery set, none of the log archive records in the history area applies, and hence the database manager cannot confirm that the current log applies.

The lack of a database archive record in the history area implies either that the database archive is very old, or that you have mounted the wrong database archive tape file. If you are intentionally restoring an old database archive, you must do a COLDLOG to avoid applying changes recorded in the current log.

Nonrecoverable Storage Pools

You can define storage pools that are not recoverable. Changes made to user data in nonrecoverable storage pools are not logged, which eliminates much of the overhead required for recovery operations described earlier in this chapter. Recovery is the responsibility of the user.

For some applications, the benefit derived from the reduced overhead far outweighs the effort of having to do your own recovery. The applications that benefit the most are those that do massive updating of a specific set of tables in the database. Such applications include:

- User programs that perform massive updates, using SQL INSERT, PUT, DELETE, and UPDATE statements.
- DBS utility DATALOAD and RELOAD operations involving thousands or millions of records.

If normal recovery procedures were in place, these applications would generate many log records. These not only cause processing overhead, but require a larger log, because the log must be large enough to hold all the records generated during the long-running LUW (along with the records of all other concurrent LUWs). Further, if you use archiving, the increased log activity causes more frequent archives.

For applications that cause excessive logging or archiving, you have two alternatives:

1. Run the application in single user mode with LOGMODE=N.
2. Place the tables that the application accesses in dbspaces that are assigned to nonrecoverable storage pools.

The first of these methods is usually preferable. For example, suppose you have an application that loads thousands of new records into an existing table. These records are the names and addresses of subscribers to a new monthly service that your company is offering. The data for new subscribers is loaded into the tables once a month. Between runs, users perform updates on the table using ISQL (for example, changing the address of an existing subscriber).

Now suppose you decide to run the application in single user mode with LOGMODE=N. The advantage is that after the application runs successfully and you create a database archive, the ISQL users have the benefit of full database recovery. The disadvantages are:

1. You must stop the application server to run in single user mode.
2. You must create a log or a database archive before running the application, and a database archive afterwards.
3. If LOGMODE is L, you lose the potential to restore the database to its current level by using a back-level database archive and subsequent log archives, because you have broken the continuity of the log archives.

Consider, though, the alternative of placing the data in a nonrecoverable storage pool. By doing so, you avoid having to create the archives, and you can run the application in multiple user mode and so avoid interrupting other users. However, the data is nonrecoverable. The decision depends on whether your ISQL or DBSU utility users can work without recovery. If the answer is no, or if you are not certain you can foresee all possible recovery situations, use LOGMODE=N instead.

Characteristics of Dbspaces in Nonrecoverable Storage Pools

The following discussion provides the basis for you to determine whether it is feasible to store the data for a given application in a dspace in a nonrecoverable storage pool, and what recovery procedures you will need for such data.

There is one situation where nonrecoverable and recoverable dbspaces have the same characteristics: when the database manager is running in single user mode with LOGMODE=N. In this situation, for both types of dbspaces, if there is a failure, all updates that were committed at the time of the failure are in the database; all those that were not committed are not. This applies to any ISQL or DBS utility command that updates the database. Note that commitment includes both an explicit COMMIT command and any implicit commitment (as described earlier in this chapter).

In any mode other than LOGMODE=N, the following characteristics apply to nonrecoverable dbspaces:

- Archiving nonrecoverable dbspaces

When you take a database archive, nonrecoverable dbspaces are archived the same way as the recoverable ones. Logging is performed differently, however, because changes to user data in nonrecoverable dbspaces are not logged.

- Locking and concurrency

Same as for recoverable dbspaces.

- Preprocessing

DB2 Server for VM preprocessors never update data in nonrecoverable dbspaces.

- Atomicity of operations

Not supported. For more information, see the discussion on the SQL statements that affect multiple rows on page 254.

- Committing work

The database manager forces a checkpoint whenever there is an implicit or explicit COMMIT of a LUW that updated data in a nonrecoverable dspace, to ensure that all updates in that LUW are really in the database. The checkpoint will only occur if data is modified, such as by an INSERT, UPDATE, or DELETE statement. It will not occur for LUWs that do not update data, or for data administration operations such as creating or dropping indexes or altering tables. These operations are logged and are thus recoverable.

Thus, except when restoring from an archive (see below), a user can be sure that committed updates are in the database, and will survive a system failure or an application failure. They do not, however, survive a DASD failure unless you archive the database after the updates are made.

Note: Checkpoints cause significant system overhead and increase response time for interactive users. Thus, avoid a high frequency of LUWs that update data in nonrecoverable dbspaces. Also, a checkpoint that occurs during a database or log archive causes the database manager to end all concurrent activity until the archive is completed, so users must wait. Plan your updates to nonrecoverable dbspaces so that they do not coincide with an archive operation.

- Rolling back work

When an LUW is rolled back (either implicitly or explicitly), the database manager does not undo successful SQL INSERT, PUT, DELETE, and UPDATE statements. Instead, it forces a checkpoint (after it rolls back any changes made to recoverable data during that LUW). This means that the nonrecoverable data appears just as though the LUW had been committed at the point when the rollback occurred.

If you want to return the data to the state it was in before the LUW, you must undo the INSERTs, PUTs, DELETES, and UPDATES manually. Until you do, other users can see the uncommitted updates.

The database manager does a checkpoint to ensure that you know what changes were made (so that you can undo them). If the checkpoint was not done, and the database manager failed before the next checkpoint, it would be difficult to tell what changes (if any) were made to the database. The checkpoint is done to make it easier for you to undo the changes.

There are two situations where the database manager does not force a checkpoint for rollbacks of LUWs that update nonrecoverable data:

- When it rolls back LUWs during a warm start after a system failure.

The database manager uses the log to determine the LUWs that were in progress at the time of the failure. These LUWs are normally rolled back. Changes to nonrecoverable data are not rolled back, because they were never recorded in the log in the first place.

There is no forced checkpoint because when the system fails, all changes made since the last checkpoint are lost. (They are not in the database.) For nonrecoverable data, in this situation, there is nothing to record at a checkpoint. For more information, see the discussion on recovering from processing failures, below.

- When it rolls back LUWs when applying log changes during an archive restore.

Here again, the updates are not in the log, so there is nothing to record at a checkpoint. In fact, all changes to nonrecoverable data made after the archive are lost. For more information, see the discussion on restoring from an archive, below.

The following consideration applies only if you have VSE guest sharing:

Usually the EXEC CICS ROLLBACK rolls back updates made to multiple resources, but the CICS transactions that use the two-phase syncpoint (TPSP) protocol cannot rely on this when nonrecoverable data is involved. You must make other provisions for such transactions.

- Recovering from processing failures

Logical units of work that are in-process when a system failure occurs lose the automatic rollback that normally is done the next time the application server is started. In this situation, the state of these updates depends on when the last checkpoint occurred before the failure. Updates that were completed before the checkpoint occurred are in the database; those done after the checkpoint are not. You must undo only the updates made by an in-process LUW that occurred before the last checkpoint. This procedure resets the data to its state before the LUW that was interrupted by the system failure.

This process applies only to nonrecoverable data. If you are also updating recoverable data in that same LUW, the normal recovery rules apply for that data.

- Restoring from an archive

If you are restoring the database from an archive copy, all data updates to nonrecoverable dbspaces done after that archive was taken are lost. You must redo all updates since the archive to bring those dbspaces to the current level.

Because row updates (INSERT, PUT, DELETE, UPDATE) are not recorded in the log, the filtered log recovery ROLLBACK COMMITTED WORK command does not apply. It does apply, however, for recoverable SQL statements and for the DBS utility command REORGANIZE INDEX (see below), because they are logged. For information about filtered log recovery, see the discussion on starting the application server to recover from a DBSS error in the *DB2 Server for VM Diagnosis Guide and Reference* manual.

- Recoverable statements and commands

The following SQL statements are always recoverable, even if they involve nonrecoverable dbspaces:

- ACQUIRE DBSPACE
- ALTER DBSPACE
- ALTER TABLE
- CREATE INDEX
- CREATE TABLE
- DROP DBSPACE
- DROP INDEX
- DROP TABLE

The DBS utility command REORGANIZE INDEX is also recoverable.

The reason these are recoverable is that the database manager does not suppress logging for them. They are logged to ensure the integrity of the database catalog tables, which always refer only to objects that exist.

If an LUW fails to commit (implicitly or explicitly) after successfully doing any of the above statements or the command, the recovery procedures will automatically undo the statement or command. For example, suppose the following actions are in a LUW:

1. CREATE TABLE
2. INSERT into that table.

If this LUW fails to be committed, the table, all its rows, and its indexes are automatically dropped from the database. Because the above statements are logged, if an LUW is committed after successfully processing the statements, they can be restored from the archive.

- Partial row updates

Except for long strings, the problem never occurs of a single row being only partially updated (inserted, deleted, or modified). The database manager always ensures that either all processing for updating a row is in the database, or that none is. (You can get partial row updates for long strings because more than one update is needed internally for each row update you request.)

- SQL statements that affect multiple rows

An SQL statement that causes multiple rows to be inserted, deleted, or updated can fail between row modifications, due to an error condition or a system failure. Whatever the cause, because the dbspace is nonrecoverable, some of the rows are modified in the database, and some are not.

Data That Can be Placed in Nonrecoverable Storage Pools

When you are considering placing application data in a nonrecoverable storage pool, you must determine whether the user will be able to recover it in a reasonable and relatively simple manner. If so, then the table is a candidate for a nonrecoverable storage pool.

Some examples of such data follow, along with descriptions of how to recover it, based on the rules in the previous section.

Example 1

Some applications use data that is retrieved from a source outside the database, such as data from another DB2 Server for VM or non-DB2 Server for VM database, or data from sequential files (including CMS files and virtual reader files). Such tables are candidates for nonrecoverable storage pools if the following are true:

1. The data, after being loaded into database tables, is used only for read-only applications.
2. The data from the outside source is the only data in the tables. (That is, the data was not added to existing tables.)

If the application that loads the data into the database tables fails (does not COMMIT) for any reason, you can recover in either of the following ways:

- If the failing LUW included one or more CREATE TABLE statements for the tables being loaded, rerun the application. Because this statement is recorded in the log, any failure to commit would cause it to be rolled back. The table and any rows that were inserted into it would be dropped.
- If the failing LUW did not contain any CREATE TABLE statements, delete all the rows from the tables; then rerun the application step that loads the data into the tables.

If, after successfully loading the data, you restore the database from a database archive that was created before the data was loaded, the rows you loaded no longer exist in the database. You can recover as follows:

1. Bypass any steps that delete all rows from the tables or that drop and recreate the tables.

These steps are not necessary because the database manager always records DROP and CREATE table statements in the log, even for nonrecoverable dbspaces.

2. Rerun the steps that load the data into the tables.

You must redo the data manipulation statements (in this situation, INSERT and PUT) because they are not recorded in the log. (The restore defined the tables in the database, but did not insert any data.)

These recovery rules apply only to data that is imported and loaded once and is discarded when no longer needed. Each time the data is loaded, it completely replaces the previous version.

The key point is that the source data must exist so that it can be used to recover the read-only database version.

Example 2

Data that is retrieved from an outside source and added to existing data can also be stored in a nonrecoverable dbspace.

The data can be from any of the sources described in Example 1 above. To add it to an existing table, you could use the DBS utility DATALOAD command or an application program to perform a mass INSERT operation.

You can recover the data if each batch of added rows has a unique value in a column that identifies rows of the batch. You would need an application program that generates a unique batch identifier and places it into each record (or into each row, if the application loads the rows into a table).

If the application that loads the data fails (does not commit the work) for any reason, you can recover as follows:

1. Specify the unique values that identify the rows added to the tables.
2. Delete all the rows in tables that have these unique identifier values. These rows were inserted before the system failed.
3. Rerun the step that loads the added data into the tables.

Note: Although it is tempting to commit work frequently during loading to avoid potential recovery problems, keep in mind that the commit operations cause checkpoints, which can adversely affect overall performance.

If you restore the database using a database archive that was created before one or more of the load operations, all rows loaded since that archive no longer exist in the database.

To recover those lost rows, either:

1. Query the tables to determine the last batch of rows inserted that still exist in the database.
2. Rerun the steps that added all subsequent batches of rows to the tables.

Alternatively:

1. Delete all the rows that were loaded before the database archive was taken of the tables.
2. Reload all of the rows from the original source.

Both methods of recovery assume that the loaded data still exists somewhere outside the database, and that each batch of rows has a unique identifier.

Example 3

Read-only data that is created by one or more INSERT via subselect statements can also be stored in a nonrecoverable dbspace. For recovery to be possible, the data must be inserted into empty tables.

If the loading of the table fails to be committed, you can recover the data as follows:

1. If the LUW created the table:
 - a. Recreate the table. (Because CREATE TABLE statements are always recoverable, the table is dropped when the LUW fails.)
 - b. Rerun the INSERT via subselect statements to load the data.
2. If the table already exists:
 - a. Delete all the rows from the table, since they reflect an incomplete update.
 - b. Rerun the INSERT via subselect statements to load the data.

If you restore the database from a database archive that was created before the data was loaded, the data that was loaded is not in the database. The table is not dropped, however, even if it was created after the archive, because CREATE TABLE statements are always logged. To restore the data that was eliminated by the database restore operation:

1. If the table was created (or recreated) after the database archive, rerun the INSERT via subselect statements.
2. If the table was created before the database archive, some rows may also exist in the table. It may be impossible to identify the INSERT via subselect statements that put these rows in the table. Even if you determine the INSERT responsible for a row, it is difficult to tell if all rows originally inserted by the statement still exist. (The statement may have been in progress at the time the database archive was taken.) For this situation:
 - a. Delete all rows in the table.
 - b. Rerun the INSERT via subselect statements.

Avoid loading (or otherwise updating) nonrecoverable dbspaces if an online database archive could occur at the same time, because such archives typically contain changes made by incomplete LUWs. For recoverable data, this is not a problem because the log contains the rest of the changes, so when you do a restore, the archive and the log are used together to reconstruct a consistent copy of the database. For nonrecoverable data however, changes are not recorded in the log, so data can be incomplete or inconsistent because no log records are available to complete the restoration of the database.

You should also not update nonrecoverable data when an online log archive can occur, because the database manager waits until all LUWs end before creating the log archive. Because LUWs that update nonrecoverable data are usually long-running, the log archive is forced to wait. If the log fills to the SLOGCUSH point, log overflow processing will be started: this involves rolling back the longest-running LUW, which is usually the one that is updating nonrecoverable

data. (For a description of the SLOGCUSH parameter, see “SLOGCUSH” on page 74.)

Data That Should Not Be Placed in Nonrecoverable Dbspaces

Any data that would be difficult or impossible for a user to recover should not be put in nonrecoverable dbspaces. Some examples are:

- Data that cannot be recreated

This includes data whose source is destroyed after the data is loaded, and data that is manually entered into tables (with the ISQL INPUT command, for example).
- Data that is modified by application programs or terminal users after it is loaded into the database

If the owner of the table keeps an audit trail of the updates made, you *can* put this kind of data in a nonrecoverable dspace, and have the owner use the audit trail to do recovery. However, this is practical only if the number of updates made is small.
- Tables that are linked with referential constraints (referential integrity) to tables in recoverable dbspaces.
- Tables that are managed by DB2 Server for VM components:
 - ISQL-stored query tables
 - ISQL-stored routine tables
 - Other IBM-supplied tables.
- Tables with small amounts of data

Here, recovery is not a problem. Rather, there is just not enough logging done for the data to justify the added complexity of user recovery. Let the database manager do the logging and recovery.
- Large tables where small numbers of rows are periodically added

Here again, there is not enough logging to justify user recovery.

Setting Up Nonrecoverable Storage Pools and Dbspaces

If you want the data for a particular application to reside in a nonrecoverable storage pool, do the following:

1. Determine the dspace requirements (size, type, and number).
2. Design a recovery scheme to use in case an LUW fails while the nonrecoverable dbspaces are being updated.
3. Design a recovery scheme to use in case restoring the database from an archive should be necessary.
4. Allocate the nonrecoverable storage pool. You can do this either during database generation, or when adding a dbextent. In either situation, use the POOL control statement (see “Adding Dbextents to a Storage Pool” on page 166).

Attention: Once a storage pool is defined, either by adding dbextents to it or by POOL(NOLOG), you must not change it from recoverable to nonrecoverable, or the reverse.

5. Define dbspaces in this storage pool, either during database generation or when adding dbspaces (see “Adding Dbspaces to the Database” on page 153). On your control statements defining the dbspaces, specify the number of the storage pool.
6. Acquire the dbspaces you want by using the ACQUIRE DBSPACE statement. You must specify the number of the storage pool you want with the STORPOOL parameter; otherwise, the database manager will not select a dbspace from a nonrecoverable storage pool.
7. Create tables in these dbspaces. To do this, you must specify the dbspace name in the CREATE TABLE statement; otherwise, the database manager will not place a table in a nonrecoverable dbspace.

Remember to perform your recovery procedures whenever there is a LUW failure or when you must restore the database from an archive.

Querying for Nonrecoverable Storage Pools and Dbspaces

To determine whether a storage pool is nonrecoverable, issue the SHOW DBEXTENT operator command. The POOL NO. column shows the number of the pool. If it is positive, the storage pool is recoverable; if negative, it is nonrecoverable. For example, if the number displayed is -32, storage pool 32 is nonrecoverable; if it is 32, this storage pool is recoverable.

To determine what dbspaces are in nonrecoverable storage pools, look at the POOL column in the SYSTEM.SYSDBSPACES catalog table. If this value is positive, the pool where the dbspace is assigned is recoverable; if it is negative, the pool is nonrecoverable. Again, the absolute value of the number is the storage pool number.

Following are some sample queries you can use to determine the status of nonrecoverable storage pools and dbspaces:

- To determine which storage pools are nonrecoverable and have dbspaces assigned to them, issue:

```
SELECT DISTINCT POOL -
FROM SYSTEM.SYSDBSPACES -
WHERE POOL > 999
```

Because the data type of the POOL column is DBAHW, you specify POOL > 999 instead of POOL < 0 to retrieve the nonrecoverable (that is, negative) storage pools. The DBAHW fields do not sort the same way that SMALLINT fields do. (See the *DB2 Server for VSE & VM SQL Reference* manual for description of data types.)

- To determine how many of the public dbspaces allocated to nonrecoverable storage pool number 7 are not yet acquired, and the number of pages in each of the dbspaces, issue:

```
SELECT NPAGES FROM SYSTEM.SYSDBSPACES -
WHERE DBSPACETYPE=1 AND POOL=-7 AND OWNER='      '
```

The blank OWNER column indicates that the dbspace is not yet acquired.

To find the same information for private dbspaces, change the DBSPACETYPE value in the statement from 1 to 2.

- To determine how many storage pools remain to be defined in the database, first issue the SHOW DBCONFIG command to see the value of the MAXPOOLS parameter. This value, which was set during database generation, determines the maximum number of storage pools allowed.

Next, issue SHOW DBEXTENT to determine the number of storage pools that are in use. Storage pools are in use only if dbextents are assigned to them. The difference between this number and MAXPOOLS is the number of pools that remain to be defined. You can define storage pools by adding extents to new pool numbers until you reach the MAXPOOLS limit.

- To determine whether a specific table is in a nonrecoverable dbspace, issue:

```
SELECT DBSPACENO FROM SYSTEM.SYSCATALOG -  
WHERE TNAME=table_name AND CREATOR=userid
```

If the DBSPACENO value is 0, the table is actually a view, and you have to query the SYSTEM.SYSVIEWS catalog table to obtain the name of the underlying table. If the DBSPACENO value is not 0, use the value in this SELECT statement:

```
SELECT POOL FROM SYSTEM.SYSDBSPACES WHERE DBSPACENO=n
```

If the returned POOL value is negative, the dbspace is nonrecoverable; if it is positive, the dbspace is recoverable.

Chapter 11. Using the Accounting Facility

The accounting facility records how resources are consumed on the database manager. Resources are consumed both by individual users, and by processes that cannot be attributed to a single user, such as startup, shutdown, checkpoints, and archives. This information is collected in fixed-length records, 80 bytes long, that describe who or what consumed resources.

The records include up to 16 bytes for installation-dependent data, where you can supply information such as account numbers or project numbers. These 16 bytes can come from:

- VM or VSE applications using VSE Guest Sharing, provided an accounting exit has been installed as described in section “Supplying Account Numbers for Users” on page 355. If VM applications use the DRDA protocol to communicate with VM servers, accounting data may still be sent.
- Applications on platforms other than VM or VSE that use the DRDA protocol to connect to DB2 Server for VM servers. In this case, 16 bytes of *user supplied data* are recorded into database manager USER accounting records. Examples of such DRDA requesters are: DB2 for OS/390 and DB2 Connect.

Note: DB2 Server for VM applications can send accounting data to DRDA servers, from which accounting records may be generated. DB2 for MVS is an example of such a server. If you already have routines to process other accounting records, you can modify them to handle the DB2 Server for VM records. You can also use the database manager itself to store your accounting data, and use ISQL to easily manipulate the data and generate reports.

Where to Find More about VM Accounting

The database manager uses VM accounting facilities. If, as you read this chapter, you need more information, refer to the *VM/ESA: Planning and Administration*, the *VM/ESA: Installation Guide*, the *VM/ESA: Planning and Administration* and the *VM/ESA: System Operation* manuals for your IBM VM System Product.

Preparing to Use the Accounting Facility

You must update the VM directory entry of each database machine that is to use accounting before you can use the accounting facility. For example, suppose the VM directory entry for a database machine contains:

```
OPTION MAXCONN 32
```

Update the statement as follows:

```
OPTION MAXCONN 32 ACCT
```

Use your normal procedures for updating the VM directory. When you add the ACCT operand, the database machine is correctly defined to use the accounting facility.

Another factor you need to consider is how the number of records that are generated will affect your current VM system accounting file. Your current procedures for handling spool-file-full conditions should handle the additional

accounting records. However, you may want to increase the value of the LIMIT operand of the CP SYSACNT macro (if you use that operand). You could also allocate more DASD spool space, or close the file more often. Initially, you should use your current procedures until you get an idea of how many accounting records are generated by the DB2 Server for VM activity at your installation.

To make a general initial estimate of how many accounting records can be generated, start the application server for normal multiple user mode access, and at the end of the day, issue the DB2 Server for VM operator commands COUNTER BEGINLUW and COUNTER CHKPOINT. The number of accounting records generated at your installation is smaller than, but proportional to, these values. That is, the database manager writes an accounting record for a user on some ends of logical units of work, and on all checkpoints. Three more accounting records are written for each run of the database manager: one for initialization, one for operation, and one for termination. You can ignore these three records when making your estimate.

For example, assume that your counters show that your installation does 2 000 logical units of work and 200 checkpoints a day. On average, these result in 1 000 accounting records generated for users (and 200 records generated for checkpoints). For environments with heavy ISQL usage, the number of records generated for users would probably be lower. Do overestimate the number of accounting records.

If you want to provide account or project numbers on the accounting records, you must code a program, as the database manager does not provide code to query users or applications for these numbers. Instead, it provides an exit to an accounting module you can modify. If you want to supply installation-dependent information, consider using an accounting exit before using the accounting facility. For more information, see “Supplying Account Numbers for Users” on page 355.

When you have updated all the database machines and have made provisions for the VM system accounting file and accounting exits, you can use the accounting facility.

Starting the Accounting Facility

To start the application server in multiple user mode to collect accounting information:

1. Log on to a database machine. (The database machine must have the ACCT operand specified on its OPTION directory control statement.)
2. If you are reconnecting, IPL CMS to get a fresh machine.
3. Start the SQLSTART EXEC with ACCOUNT=D. You can set all other initialization parameters as you would usually set them. For example:

```
SQLSTART DB(DB010) ID(B00T1) PARM(PARMID=WARM,NLRBU=1500,ACCOUNT=D)
```

The file WARM SQLPARM is referenced in the PARMID parameter. The NLRBU and ACCOUNT parameters are specified on the command itself. The ACCOUNT parameter is a normal initialization parameter, so you can specify it in a CMS parameter file.

The database manager then automatically generates accounting records for all activity involving the database machine.

The database manager can also generate accounting records in single user mode for user programs, the DBS utility, and the preprocessors. Accounting records are not generated for:

- Log reconfigurations (COLDLOG) with the SQLLOG EXEC. The SQLLOG EXEC starts the application server with STARTUP=L.
- Database generations with the SQLDBGEN or SQLDBINS EXECs. These EXECs start the application server with STARTUP=C.
- Adding dbextents with the SQLADBEX EXEC. The SQLADBEX EXEC starts the application server with STARTUP=E.
- Adding dbspaces with the SQLADBSP EXEC. The SQLADBSP EXEC starts the application server with STARTUP=S.
- Catalog index reorganizations with the SQLCIREO EXEC. The SQLCIREO EXEC starts the application server with STARTUP=I.
- Catalog migrations with the ARISMEX EXEC. The ARISMEX EXEC starts the application server with STARTUP=M.
- PROGNAM=ARISEGB, which is the catalog update phase of an ADD DBSPACE operation.

If you specify ACCOUNT=D in any of those situations, the database manager displays a warning message and ignores the ACCOUNT parameter.

To generate accounting records for single user mode programs, specify ACCOUNT=D as you would for multiple user mode. For the DBS utility and the preprocessors, you must use the PARMID(*filename*) parameter to point to a CMS file that contains the ACCOUNT=D parameter.

Generation of Accounting Records

Accounting records are written when one of the following occurs:

- An IUCV or APPC/VM SEVER occurs:
 - When the RELEASE option of an SQL COMMIT or ROLLBACK command is specified in multiple user mode
 - When SQLHX is entered

The SQLHX command causes a rollback, which causes an accounting record to be written for the work done to that point. If no additional database work is done, and the application exits, COMMIT WORK or ROLLBACK WORK processing occurs, and a second accounting record is written. Because no work is done on the database for the second accounting record, the following fields may be empty:

- Package name (bytes 53–60 filled with blanks)
 - Active time (bytes 61–64 initialized to 0)
 - Number of looks (bytes 69–72 initialized to 0).
- When an HX of the user command is done

- When the user ID abends
- When the DB2 Server for VM operator issues an SQL FORCE for the authorization id
- When the VM operator issues a CP FORCE for the VM user ID
- A user reconnects without explicitly releasing the previous session.
For example, suppose user ID USER1 uses ISQL to implicitly connect to the database manager, does some work, and then explicitly connects as authorization ID SQLDBA to do tasks requiring DBA authority. When USER1 changes authorization IDs, the database manager writes an accounting record for authorization ID USER1 and begins a new session for authorization ID SQLDBA, even though USER1 did not explicitly release the first connection.
- The internal DB2 Server for VM resource threshold is met or exceeded. This is checked at the end of a logical unit of work.
For example, suppose USER1 uses ISQL with AUTOCOMMIT ON, and never issues a COMMIT or ROLLBACK WORK RELEASE. The session therefore lasts until this user reconnects or leaves ISQL. If this user works on ISQL for hours and processes many logical units of work during this long session, he or she exceeds the resource threshold a number of times. Every time this happens, an accounting record is written. Now suppose the database manager abends. The only accounting information lost is for work that USER1 did after last exceeding the threshold. If the internal threshold were not used, all accounting information about USER1's session would have been lost, which represents a significant amount of work.
- The connection between the user ID and the database manager is ended by:
 - An SQLEND QUICK command
 - A DB2 Server for VM FORCE command
 - A CP FORCE command.

The database manager does not write an accounting record for every logical unit of work, because too many records would be generated, resulting in high system overhead. Because most ISQL users use AUTOCOMMIT ON, practically every SQL statement issued would cause a new LUW.

If your users are using DRDA protocol, the database manager also generates accounting records for them in addition to the CMS user record. For a description of the accounting records, see "Remote User Records" on page 269.

If you have VSE guest sharing, the database manager also generates accounting records for VSE guest users. These records are similar in format to the accounting records for DB2 Server for VM users on VM. The VSE guest user records are generated on the database machine. For a description of the records, see "VSE Guest User Records" on page 269.

The database manager processes batch/ICCF users on VSE as individual users. All CICS users are processed as one user, and are identified by the user ID that CICS uses.

Supplying Accounting Data from DRDA Applications

Remote DRDA application requesters have the opportunity to send accounting information to DRDA servers using a general purpose unarchitected DRDA parameter. DB2 for MVS (Version 2 Release 3 or later) and DDCS (Version 2 Release 1) have implemented this approach for sending accounting data. Similar support was enabled for VM requesters in Version 3 Release 5.

If the database manager determines that a DRDA requester has supplied accounting data, 16 bytes of *user supplied data* is recorded into database manager USER accounting records as “installation-dependent” data. For DB2 for MVS applications, user supplied data corresponds to the MVS accounting string associated with the DB2 SQL application's MVS address space.

For DDCS applications and DB2 CONNECT applications, user supplied data corresponds to one of the following:

- The value specified by an application with the *sqlsact()* API
- The value of the *DB2ACCOUNT* environment variable
- The value of the *DFT_ACCOUNT_STR* (default accounting string) configuration parameter.

If the DRDA protocol is used to connect VM applications to VSE servers (or any other DRDA server), user supplied data corresponds to data supplied by the ARIUXIT accounting exit described in the *DB2 Server for VM System Administration* manual.

If the database manager determines that a DRDA requester has supplied accounting data but the requester is not DB2 for MVS, DDCS or DB2 CONNECT or DB2 for VM, it inserts the string “*pppvrrm UNKNOWN*” into USER accounting records. *pppvrrm* is the product id (prdid) of the DRDA requester.

Note: When you are using the DRDA protocol, the installation-dependent data should conform to the following:

1. The accounting string data is converted to CCSID 500 before being sent to the DRDA server. To ensure that all characters in the string data can be represented in CCSID 500, only the characters A-Z, 0-9 and '_' (underscore) be used. If characters other than these recommended ones are used, then those characters may not translate properly when the DRDA server writes out accounting records.
2. The user-specified portion of the accounting string can be at most 16 bytes. This is true for DB2 Server for VM applications sending accounting data (which is set up in the ARIUXIT user exit) and for non-DB2 for VM DRDA requesters sending accounting data to servers.

Formats of the Accounting Records

There are three kinds of accounting records generated for users:

CMS user records

are generated for users on VM who access an application server on VM.

Remote user records

are generated if your users are using DRDA protocol. The database manager generates accounting records for them, in addition to the CMS user records.

VSE guest user records

are generated for users on VSE who access an application server on a VM operating systems.

Accounting records are also generated for system processes that cannot be attributed to a single user:

An initialization

record is written when the application server is started. This record describes the resources consumed by the operator and checkpoint agents during the startup process.

A checkpoint

record is written for the checkpoint agent after a checkpoint occurs. For the checkpoint that immediately follows an archive, this record reflects the resources consumed in doing the archive as well as the checkpoint.

An operation

record is written during shutdown for the processing that the operator agent has done during the current session. (This accounting record is written only for multiple user mode, as operator communications are not possible in single user mode.)

A termination

record is written that summarizes the resources consumed during the current session.

Note: Internal resource thresholds are not used for system processes.

Initialization Records

Columns:

1	9	17	25	41	53	57	61	65	69	73	75	79
SQLDBA	SQL/DS	INIT		051389182005						19	ISQLC0	

Column	Data Type	Description
1-8	CHAR (8)	VM user ID of the database machine (fixed by CP)
9-16	CHAR (8)	"SQL/DS "
17-24	CHAR (8)	"INIT "
25-40	CHAR (16)	Reserved (blanks)
41-52	CHAR (12)	Date and time of the accounting record (MMDDYYHHMMSS)
53-56	CHAR (4)	Blank
57-60	CHAR (4)	Blank
61-64	INTEGER	Duration of the startup process (in seconds)
65-68	INTEGER	Processor time used by the startup process (in milliseconds)

Column	Data Type	Description
69-72	INTEGER	Number of times the database manager looked at a page buffer during startup (equivalent to issuing COUNTER LPAGBUFF immediately after startup)
73-74	CHAR (2)	Century number of Date ('19' or '20')
75-78	CHAR (4)	The xSQL identifier to separate DB2 Server for VM accounting records from other VM accounting records, where x = I for Initialization
79-80	CHAR (2)	Record identifier (character X'C0') fixed by CP

Operator and Checkpoint Records

Columns:

```

1      9      17      25      41      53  57  61  65  69  73  75  79
|      |      |      |      |      |  |  |  |  |  |  |
SQLDBA SQL/DS SYSTEM          051389182005          0083032819 CSQLC0

```

Column	Data Type	Description
1-8	CHAR (8)	VM user ID of the database machine (fixed by CP)
9-16	CHAR (8)	"SQL/DS "
17-24	CHAR (8)	"SYSTEM "
25-40	CHAR (16)	Reserved (blanks)
41-52	CHAR (12)	Date and time of the accounting record (MMDDYYHHMMSS)
53-56	CHAR (4)	Blank
57-60	CHAR (4)	Blank
61-64	INTEGER	Binary zero
65-68	INTEGER	Processor time used (in milliseconds)
69-72	INTEGER	Number of times this agent looked at a page buffer (equivalent to issuing COUNTER LPAGBUFF for only this agent)
73-74	CHAR (2)	Century number of Date ('19' or '20')
75-78	CHAR (4)	The xSQL identifier to separate DB2 Server for VM accounting records from other VM accounting records, where x = C for Checkpoint or O for Operator).
79-80	CHAR (2)	Record identifier (character X'C0') fixed by CP

Termination Records

Columns:

```

1      9      17      25      41      53  57  61  65  69  73  75  79
|      |      |      |      |      |  |  |  |  |  |  |
SQLDBA SQL/DS TERM          051389182005          19 TSQLC0

```

Column	Data Type	Description
1-8	CHAR (8)	VM user ID of the database machine (fixed by CP)

Column	Data Type	Description
9-16	CHAR (8)	"SQL/DS "
17-24	CHAR (8)	"TERM "
25-40	CHAR (16)	Reserved (blanks)
41-52	CHAR (12)	Date and time of the accounting record (MMDDYYHHMMSS)
53-56	CHAR (4)	Blank
57-60	CHAR (4)	Blank
61-64	INTEGER	Time, in seconds, from startup to shutdown
Note: The following are totals for the entire run of the database manager that are extracted from the data that is used by the COUNTER command.		
65-68	INTEGER	DASDIO - Total number of DASD I/Os
69-72	INTEGER	LPAGBUFF - Number of times the database manager looked at a page buffer
73-74	CHAR (2)	Century number of Date ('19' or '20')
75-78	CHAR (4)	The xSQL identifier to separate DB2 Server for VM accounting records from other VM accounting records, where x = T for Termination
79-80	CHAR (2)	Record identifier (character X'C0') fixed by CP

CMS User Records

Columns:

```

1      9      17      25      41      53 57 61 65 69 73 75 79
|      |      |      |      |      |  |  |  |  |  |  |  |
SQLDBA JESSICA MYID A1015TEST 051389182005BADDEBTS      19 USQLC0

```

Column	Data Type	Description
1-8	CHAR (8)	VM user ID of the database machine (fixed by CP)
9-16	CHAR (8)	VM user ID of the user machine accessing the application server
17-24	CHAR (8)	DB2 Server for VM authorization ID that was established, implicitly or explicitly, using the connect process. Unless the user explicitly changed the connected user ID, the DB2 Server for VM user ID is the same as the VM user ID.
25-40	CHAR (16)	If you have coded your own ARIUXIT exit to generate installation-supplied data, this data is placed here for CMS applications. If you have not coded such as exit, this contains character blanks for CMS applications. For non-VM DRDA applications, up to 16 bytes of user-supplied data is put here. For more information, see "Supplying Accounting Data from DRDA Applications" on page 265.
41-52	CHAR (12)	Date and time of the accounting record (MMDDYYHHMMSS)
53-60	CHAR (8)	The name of the package that was last active for the application
Note: The following are totals for the agent. They show values accumulated for a user.		
61-64	INTEGER	Active time (that is, time that the user was connected to an agent) in seconds
65-68	INTEGER	Processor time used (in milliseconds)
69-72	INTEGER	Number of times this agent looked at a page buffer (this value is equivalent to the LPAGBUFF counter value for an individual user)

Column	Data Type	Description
73-74	CHAR (2)	Century number of Date ('19' or '20')
75-78	CHAR (4)	The xSQL identifier to separate DB2 Server for VM accounting records from other VM accounting records, where x = U for User
79-80	CHAR (2)	Record identifier (character X'C0') fixed by CP

Remote User Records

Columns:

```

1      9      17      25      37      64      73 75 79
|      |      |      |      |      |      | | |
SQLDBA JESSICA MYID 051389182005nnTORONET.SP6AGATnnnnnnnn 19RSQLC0

```

Column	Data Type	Description
1-8	CHAR (8)	VM user ID of the database machine (application server)
9-16	CHAR (8)	Access user ID of the application or interactive user (application requester) accessing the application server
17-24	CHAR (8)	DB2 Server for VM authorization ID that was established, implicitly or explicitly, using the connect process
25-36	CHAR (12)	Date and time of the accounting record (MMDDYYHHMMSS)
37-63	CHAR(27)	<p>LU 6.2 LUWID. This field is composed of the following subfields:</p> <p>37-37 Length of the entire LUWID: a 1-byte binary integer</p> <p>38-38 Length of the qualified LUNAME: a 1-byte binary integer</p> <p>39-<i>n</i> Qualified LUNAME (NETID.LUNAME): a character subfield in which <i>n</i> depends on the length value in column 38</p> <p>(<i>n</i>+1)-(<i>n</i>+6) Instance number: a bit data field</p> <p>(<i>n</i>+7)-(<i>n</i>+8) Sequence number: a bit data field</p> <p>If the LUWID is less than 25 bytes, the remaining columns are padded with blanks</p>
64-72		Reserved
73-74	CHAR (2)	Century number of Date ('19' or '20')
75-78	CHAR (4)	The xSQL identifier to separate the DB2 Server for VM accounting records from other VM accounting records, where x = R for remote user
79-80	CHAR (2)	Record identifier (character 'C0') fixed by CP

VSE Guest User Records

Columns:

```

1      9      17      25      41      53 57 61 65 69 73 75 79
|      |      |      |      |      | | | | | | |
SQLDBA VSEMCH1 MYID USER DATA HERE 051389182005DEBTS 19USQLC0

```

Column	Data Type	Description
1-8	CHAR (8)	VM user ID of the database machine (fixed by CP)
9-16	CHAR (8)	For batch and VSE/ICCF environments, the jobname of the user partition. For online environments, the VM user ID of the VSE machine. (The example record above is for online environments.)
17-24	CHAR (8)	DB2 Server for VM connected authorization ID that was established using the connect process (this can be an explicit or implicit connection)
25-40	CHAR (16)	Installation-supplied data. If you are in a batch or VSE/ICCF environment, and have not coded an accounting exit that supplies information to this field, the database manager leaves character blanks. In an online environment, if you have not coded an accounting exit to supply the information, the following is put in the field: 25-28 CICS transaction ID 29-31 CICS terminal operator ID (if available) 32-35 CICS terminal ID (if available) 36-39 This field contains character blanks, unless you have coded your own cancel exit. For information on cancel exits in VSE, see the <i>DB2 Server for VM Diagnosis Guide and Reference</i> manual. 40 Blank
41-52	CHAR (12)	Date and time of the accounting record (MMDDYYHHMMSS)
53-60	CHAR (8)	The name of the package that was last active for the application (also referred to as prepname or program name)
Note: The following are totals for the agent. They show values accumulated for a user.		
61-64	INTEGER	Active time (the time that the user was connected to an agent) in seconds
65-68	INTEGER	Processor time used (in milliseconds). In the VSE guest user accounting record passed to VM/ESA, processor time is recorded in thousandths of a second (milliseconds).
69-72	INTEGER	Number of times this agent looked at a page buffer (equivalent to the LPAGBUFF counter value for an individual user)
73-74	CHAR (2)	Century number of Date ('19' or '20')
75-78	CHAR (4)	The xSQL identifier to separate DB2 Server for VM accounting records from other VM accounting records, where x = U for User.
79-80	CHAR (2)	Record identifier (character X'C0') fixed by CP

Maintaining Accounting Data

Accounting data, like any other data, can be loaded into tables and maintained by any DB2 Server for VM facility. The following sections describe how to set up dbspaces to hold accounting records and present an example. You will have to modify the example tables to meet your own installation's requirements.

Setting up a database for accounting data involves the same activities that would be done for any data application:

1. Adding and acquiring a dspace
2. Creating tables for the accounting data
3. Creating views on those tables
4. Creating indexes on those tables.

Considerations for an Accounting Dbspace

Because accounting data is usually read-only, it is most suited for a private dbspace. When it is in a private dbspace, multiple users are able to read it as long as the tables are not being loaded. (If they are being loaded, users get an immediate notification that a load is taking place in the form of a negative SQLCODE).

Also, because the data is read-only and because its source is a sequential file, it is a candidate for a nonrecoverable dbspace. For information on the advantages and disadvantages of this type of storage, see “Nonrecoverable Storage Pools” on page 250.

The size of the dbspace depends on a number of factors. The key considerations are:

- The number of accounting records you want to keep online
- The row length of the records
- The index space requirements.

When you have determined these factors, you can estimate the size of the dbspace needed by using the formulas in Appendix B, “Estimating Database Storage” on page 449.

To estimate the rate at which your installation generates accounting records, use the accounting facility for a trial period (a day or a week). Or, you can try to make an initial estimate using the method shown on page 262.

Tables to Hold Accounting Data

One approach to organizing accounting records is to place them in four separate tables:

- One to hold the termination records, which summarize the resources consumed during an entire session of the database manager.
- One to hold the initialization, operator, and checkpoint records, which describe the overhead resources consumed by the database manager processes.
- One to hold user records, which describe the resources consumed by individual users.
- One to hold remote access records, which contain the LUWID. The records also contain the user ID and datetime value that can be used to match with the regular user records.

Figure 76 shows the statements you could issue to create these four tables, here named SQLDETAIL (for termination records), SYSDETAIL (for initialization, operator and checkpoint records), USERDETAIL (for user records), and REMDETAIL (for remote user records).

```

CREATE TABLE SQLDETAIL(SQLNAME CHAR(8),
                        DATE CHAR(6),
                        TIME CHAR(6),
                        RUNTIME INTEGER,
                        DASDIO INTEGER,
                        LPAGBUFF INTEGER,
                        CENTURY CHAR(2) ) IN SQLDBA.ACCTNG;

CREATE TABLE SYSDetail(SQLNAME CHAR(8),
                        TYPE CHAR(8),
                        DATE CHAR(6),
                        TIME CHAR(6),
                        RUNTIME INTEGER,
                        CPUTIME INTEGER,
                        LPAGBUFF INTEGER,
                        CENTURY CHAR(2) ) IN SQLDBA.ACCTNG;

CREATE TABLE USERDETAIL(SQLNAME CHAR(8),
                         CPUSER CHAR(8),
                         SQLUSER CHAR(8),
                         USERDATA CHAR(16),
                         DATE CHAR(6),
                         TIME CHAR(6),
                         PNAME CHAR(8),
                         ATIME INTEGER,
                         CPUTIME INTEGER,
                         ULPAGBUF INTEGER,
                         CENTURY CHAR(2) ) IN SQLDBA.ACCTNG;

CREATE TABLE REMDETAIL(SQLNAME CHAR(8),
                        CPUSER CHAR(8),
                        SQLUSER CHAR(8),
                        DATE CHAR(6),
                        TIME CHAR(6),
                        LUWID VARCHAR(27),
                        CENTURY CHAR(2) ) IN SQLDBA.ACCTNG;

```

Figure 76. Example of DBS Utility Commands to Create Accounting Tables

Note: If you have accounting tables defined from an earlier release, you can use the ALTER TABLE statement to add the CENTURY column to your existing tables.

The information for all the columns in the tables is loaded directly from the accounting records. These tables are described in detail below.

SQLDETAIL Table

Each row of the SQLDETAIL table contains selected data from one termination accounting record, and represents one session of the database manager. The following information is inserted into the SQLDETAIL columns:

- SQLNAME** The VM user ID of the database machine
- DATE** The dates from the termination records
- TIME** The times from the termination records
- RUNTIME** The time, in seconds, from startup to shutdown

DASDIO The total number of DASD I/Os for the database manager session

LPAGBUFF The total number of times that the database manager looked at a page buffer

CENTURY The century numbers of the dates from the termination records.

SYSDetail Table

Each row of the SYSDetail table contains selected data from one initialization, operator or checkpoint accounting record. The following information is inserted into its columns:

SQLNAME The VM user ID of the database machine

TYPE INIT is inserted if the row describes an initialization record, and SYSTEM is inserted if the row describes an operator or checkpoint record

DATE The dates from the operator/checkpoint or initialization records

TIME The times from the operator/checkpoint or initialization records

RUNTIME If the value in TYPE is INIT, this value shows the amount of time for the initialization process to finish (in seconds); if the value in TYPE is SYSTEM, this value contains binary zeros

CPUTIME This column contains the processor time used (in milliseconds)

LPAGBUFF The number of times the agent (represented by the accounting record) looked into a page buffer

CENTURY The century numbers of the dates from the initialization records.

USERDetail Table

Each row of the USERDetail table contains selected data from one user accounting record, either from a local or a remote processor. The following information is inserted into its columns:

SQLNAME The VM user ID of the database machine

CPUSER The VM user ID of the user machine accessing the application server (for VSE guests in batch and ICCF environments, it contains the job name of the user partition)

SQLUSER The authorization ID that was established, explicitly or implicitly, during the connect process

USERDATA The installation-supplied accounting data. If no accounting exit was coded to supply installation-dependent data, you can omit this column when defining the table. If you use the VM/ESA operating system, the first 4 bytes of this column contain the CMS work unit ID. If you have your own accounting exit, and it uses the first 4 bytes of this column, it will write over the CMS work unit ID. If you have VSE guest sharing, this column contains the following information for online environments:

- The CICS transaction ID
- The CICS terminal operator ID (if available)
- The CICS terminal ID (if available)
- Any data you supply through your own cancel exit.

DATE The dates from the user records

TIME The times from the user records

PNAME The name of the package that was last active for the application

ATIME The active time (that is, the time that the user was connected to an agent) in seconds

CPUTIME The processor time used (in milliseconds)

ULPAGBUF The number of times the agent looked into a page buffer.

CENTURY The century numbers of the dates from the user records.

REMDetail Table

Each row of the REMDETAIL table contains selected data from one remote user accounting record. The columns are described as follows:

SQLNAME The VM user ID of the database machine (application server)

CPUSER The VM user ID of the user machine accessing the application server (for VSE guests in batch and ICFE environments, the job name of the user partition)

SQLUSER The authorization ID that was established, explicitly or implicitly, during the connect process

DATE The dates from the remote user records

TIME The times from the remote user records

LUID The qualified LUNAME, the sequence number, and the instance number

CENTURY The century numbers of the dates from the remote user records.

Loading the Accounting Data

If you have created the tables described above, you can use the DBS utility to load the accounting records into the tables, as shown in Figure 77.

The ARIACC1 file contains the accounting records. The file can be the VM system accounting file. The DBS utility selects the DB2 Server for VM records from the file by using positions 75-78 to identify the records.

```
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
SET ERRORMODE CONTINUE;
DATALOAD TABLE(SQLDETAIL) IF POS(75-78)='TSQL'
  SQLNAME 1-8 CHAR
  DATE    41-46 CHAR
  TIME    47-52 CHAR
  RUNTIME 61-64 FIXED
  DASDIO  65-68 FIXED
  LPAGBUFF 69-72 FIXED
  CENTURY 73-74 CHAR NULL IF POS(73-74) = 0
```

Figure 77 (Part 1 of 3). Example DBS Utility Commands to Load Accounting Tables

```

DATALOAD TABLE(SYSDetail) IF POS(75-78)='ISQL'
  SQLNAME  1-8  CHAR
  TYPE     17-24 CHAR
  DATE     41-46 CHAR
  TIME     47-52 CHAR
  RUNTIME  61-64 FIXED
  CPUTIME  65-68 FIXED
  LPAGBUFF 69-72 FIXED
  CENTURY  73-74 CHAR  NULL IF POS(73-74) = 0

DATALOAD TABLE(SYSDetail) IF POS(75-78)='OSQL'
  SQLNAME  1-8  CHAR
  TYPE     17-24 CHAR
  DATE     41-46 CHAR
  TIME     47-52 CHAR
  RUNTIME  61-64 FIXED
  CPUTIME  65-68 FIXED
  LPAGBUFF 69-72 FIXED
  CENTURY  73-74 CHAR  NULL IF POS(73-74) = 0

DATALOAD TABLE(SYSDetail) IF POS(75-78)='CSQL'
  SQLNAME  1-8  CHAR
  TYPE     17-24 CHAR
  DATE     41-46 CHAR
  TIME     47-52 CHAR
  RUNTIME  61-64 FIXED
  CPUTIME  65-68 FIXED
  LPAGBUFF 69-72 FIXED
  CENTURY  73-74 CHAR  NULL IF POS(73-74) = 0

DATALOAD TABLE(USERDetail) IF POS(75-78)='USQL'
  SQLNAME  1-8  CHAR
  CPUSER   9-16 CHAR
  SQLUSER  17-24 CHAR
  USERDATA 25-40 CHAR
  DATE     41-46 CHAR
  TIME     47-52 CHAR
  PNAME    53-60 CHAR
  ATIME    61-64 FIXED
  CPUTIME  65-68 FIXED
  ULPAGBUF 69-72 FIXED
  CENTURY  73-74 CHAR  NULL IF POS(73-74) = 0

DATALOAD TABLE(REMDetail) IF POS(75-78)='RSQL'
  SQLNAME  1-8  CHAR
  CPUSER   9-16 CHAR
  SQLUSER  17-24 CHAR
  DATE     25-30 CHAR
  TIME     31-36 CHAR
  LUWID    37-63 CHAR
  CENTURY  73-74 CHAR  NULL IF POS(73-74) = 0

```

Figure 77 (Part 2 of 3). Example DBS Utility Commands to Load Accounting Tables

```
INFILE(ARIACC1)
COMMIT WORK;
SELECT * FROM SQLDETAIL;
SELECT * FROM SYSDETAIL;
SELECT * FROM USERDETAIL;
SELECT * FROM REMDETAIL;
```

Figure 77 (Part 3 of 3). Example DBS Utility Commands to Load Accounting Tables

Chapter 12. Planning and Implementing Configurations

Initially, you set up one database machine with one database: then, depending on your needs, you can add additional databases or database machines. If you use a VM/ESA operating system and you have VSE running as a guest, you may also want to set up VSE guest sharing.

The chapter presents the following topics:

- Configuration concepts

This section discusses the reasons for having more than one database machine, and what this means to the user; database machines in a TSAF collection or in an SNA network, and what this means to the user; adding service machines, and considerations for national language support; types of database machines; and environment considerations.

- Primary database machines

This section discusses the reason for adding a primary database machine, and shows examples of adding a primary and a secondary database machine.

- Adding a service machine
- Defining additional user machines
- Adding a database

This section contains an example of adding a database, and describes the options that you can specify for it and how to change the password of user ID SQLDBA.

- VSE guest sharing

This section presents general information on VSE guest sharing, restrictions on guest sharing, and examples of guest sharing configurations.

Configuration Concepts

This section discusses the following topics:

- The reasons for adding an additional database machine
- Databases in a TSAF collection or an SNA network
- Adding service machines
- Types of database machines.

Reasons for Adding a Database Machine

Initially, a set-up contains one database machine which is called SQLMACH. As your installation grows, you can add more database machines. The primary reason for doing so would be to permit multiple user mode access to more than one database at the same time, or *multiple database operation*.

Consider, for example, an installation having one database machine (SQLMACH) and three databases (SQLDBA, DATA1, and DATA2). A database machine can access only one database at a time. Thus, as Figure 78 shows, while the database machine is accessing one database in multiple user mode, the other databases are inactive.

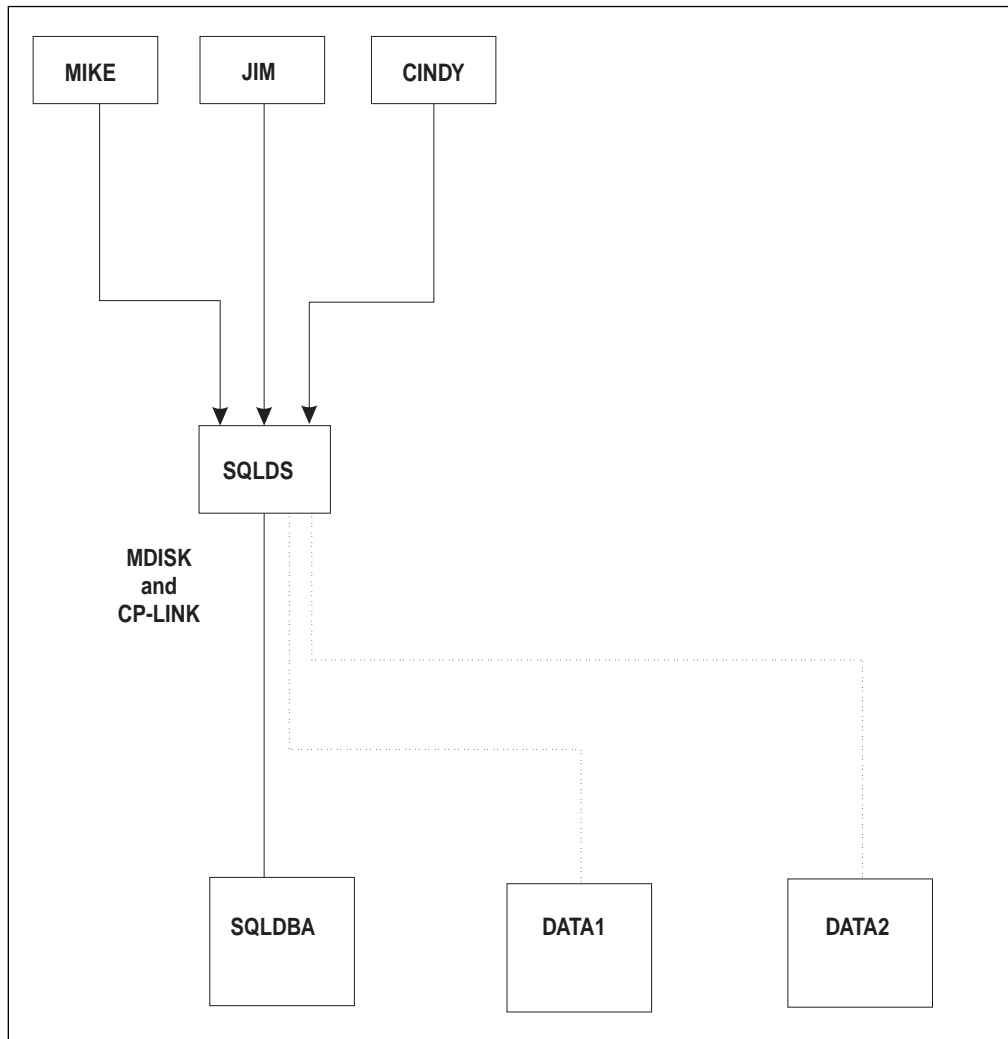


Figure 78. One Database Machine Accessing One Database

Users could access the remaining databases (DATA1 and DATA2) in single user mode if their machines were set up to do so; however, it is not recommended that the database manager be used this way.

If you define two more database machines, multiple user access to all three databases is now possible at the same time. Figure 79 shows a multiple database configuration. In this case, two more database machines are defined (SQLMFB and SQLJDS). Each machine “owns” one database (that is, has the MDISK entries in its directory for the database minidisks), and operates independently of the others.

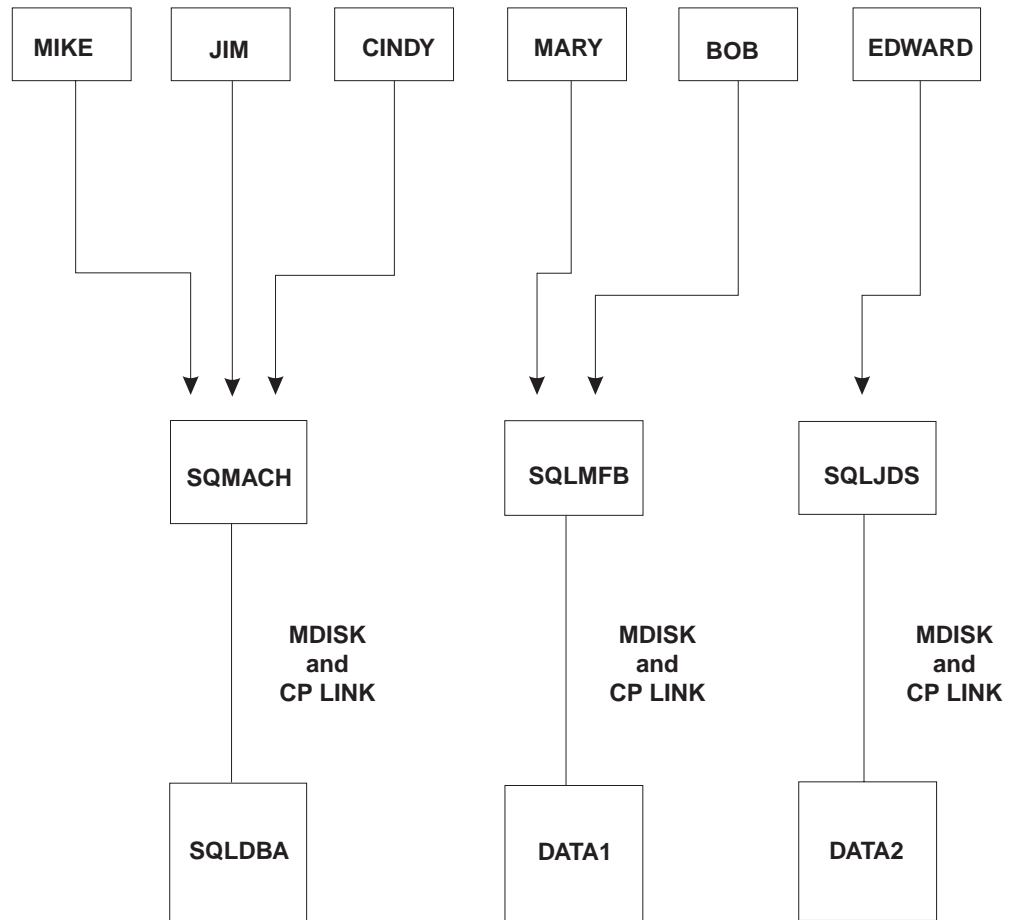


Figure 79. Multiple Users Accessing Multiple Databases

Implications for Users

Although database machines operate independently of each other, users are not restricted to a particular one.

Suppose user CINDY in Figure 79 has CONNECT authority on all three databases. With this configuration, she can move between databases by using the SQLINIT EXEC or the SQL CONNECT statement.

For example, CINDY could issue SQLINIT DBNAME(SQLDBA) to prepare her virtual machine for accessing the SQLDBA database, and then use ISQL to access that database. To access any of the other databases, she could leave ISQL, reissue the SQLINIT EXEC (specifying another database), and then access that database using ISQL. She could issue the SQL CONNECT statement to switch databases without leaving ISQL. (The SQL CONNECT statement is described in the *DB2 Server for VM Database Administration* manual.)

Databases in a TSAF Collection or an SNA Network

Processors can be part of either a TSAF collection or an SNA network.

The TSAF collection is composed of a group of processors with the TSAF virtual machine component installed, running and connected. A *distributed relational database network* is a group of interconnected processors supporting relational databases that can be accessed locally or remotely through either the SQLDS protocol or the DRDA protocol. Each processor gains access to the SNA network through its communications subsystem. On the database manager, the communications subsystem consists of the CMS communications directory, APPC/VM, TSAF, AVS, and the VTAM product.

All databases generated on these processor must be identified as either LOCAL or GLOBAL resources. A LOCAL database can be accessed only by users located on the same processor as the database. A GLOBAL database can be accessed by users located on other processors as well, as long as the processor on which the database resides is physically connected to those processors or is in a TSAF collection.

A DB2 Server for VM application server can receive communications from other relational database products when the PROTOCOL=AUTO parameter of the SQLSTART EXEC is specified.

A DB2 Server for VM application requester can communicate with other relational database products when either DRDA or AUTO is specified for the PROTOCOL parameter of the SQLINIT EXEC.

If the application requester uses DRDA protocol to communicate with a like application server, the overhead increases due to the protocol conversion between the two. The use of DRDA protocol between like machines is usually only for problem determination or modelling.

In Figure 80, application servers SQLA and SQLB are on two processors connected in a TSAF collection. One of the processors has access to the SNA network through AVS. The application requester USER1 is on the other processor. It is assumed that the application server SQLB supports DRDA protocol.

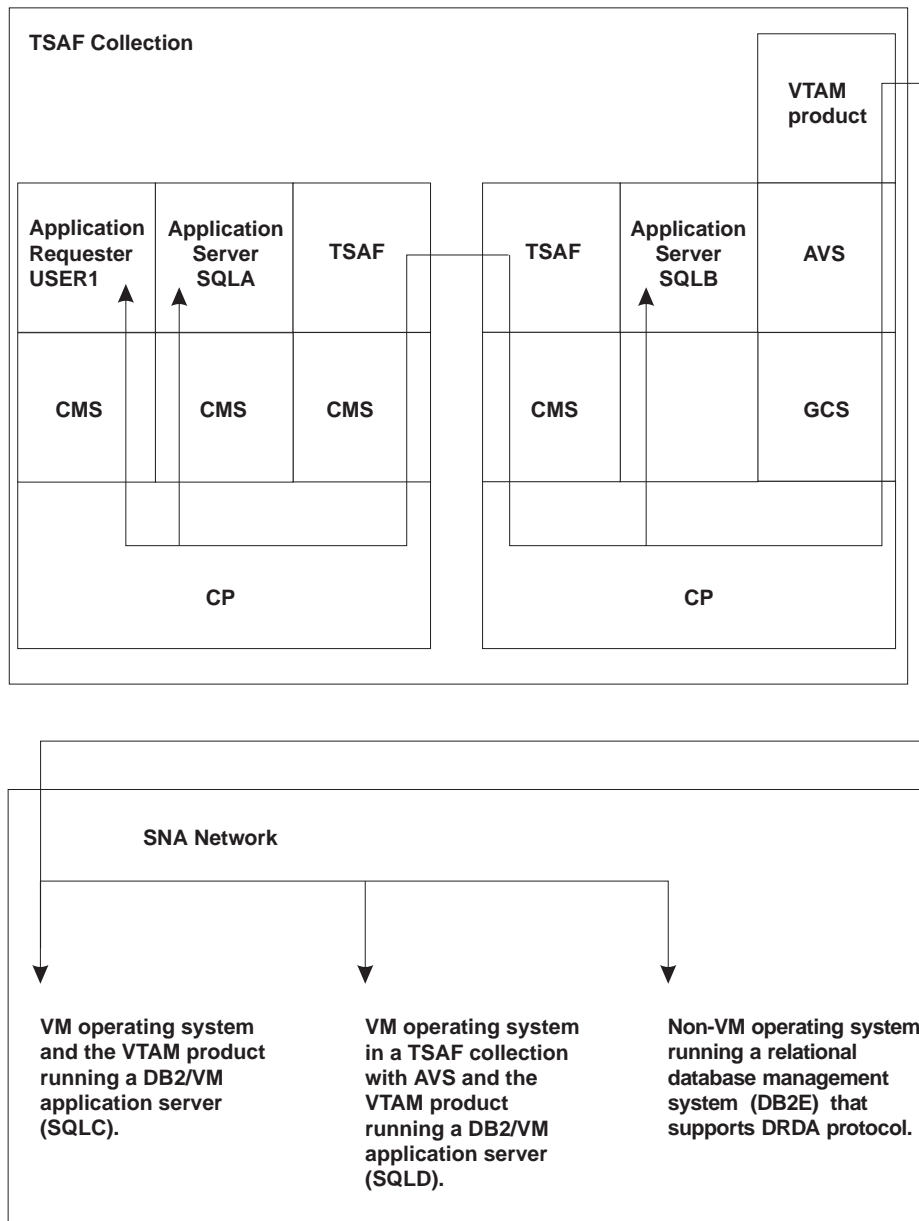


Figure 80. Accessing Application Servers in a Network

Application server SQLA is LOCAL; thus it can be accessed only by application requesters that are on the same processor, such as USER1. Application server SQLB is GLOBAL, so it can be accessed by application requesters such as USER1 located anywhere in the TSAF collection or SNA network. Because application server SQLB supports the DRDA protocol, it can also be accessed by unlike application requesters that support the DRDA protocol in the SNA network.

Application requester USER1 can access DB2 Server for VM application servers in both the TSAF collection and the SNA network, as well as unlike application servers that support the DRDA protocol in the SNA network.

Implications for the User

Users do not have to know the location of the application servers: all they need to know is the server name (*server_name*). For example, in Figure 80, a user can establish a default application server for implied connection by entering one of the following SQLINIT EXEC commands:

```
SQLINIT DBNAME(SQLA)
SQLINIT DBNAME(SQLB)
SQLINIT DBNAME(SQLC)
```

The user then can access the default application server through ISQL, the DBS utility, or an application program, or can issue the SQL CONNECT statement to switch to another application server. For example, if USER1 accesses the SQLA application server through ISQL, he or she can later switch to the SQLC application server, without having to exit ISQL, by entering CONNECT TO SQLC. For more information on the SQL CONNECT statement, see the *DB2 Server for VM Database Administration* manual.

AVS Session Limit Considerations

When an application requester uses AVS to communicate with a remote application server, a connection is initiated. If this connection causes the established session limit to be exceeded, AVS puts the connection in a pending state. This state is maintained indefinitely until another session becomes available, which happens when an existing connection is disconnected or terminated. AVS then allocates the connection on this session, and control is returned to the user application.

Attention: To avoid this situation, it is recommended to plan for peak usage. The calculated session limit should be increased to allow for some additional connections.

Adding Service Machines

If users do not have a database machine installed on their processors, they can still access database machines on other processors if a service machine has been installed. In a TSAF collection a service machine gives users access to a production minidisk that contains DB2 Server for VM files, such as SQLINIT EXEC and ISQL EXEC, necessary for users to access the databases.

Figure 81 shows an example of a user accessing an application server located on another processor in a TSAF collection.

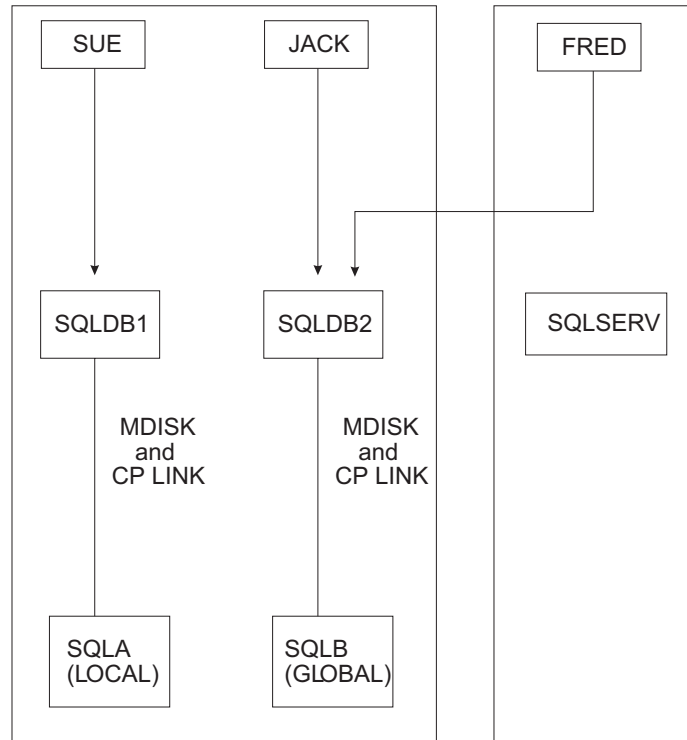


Figure 81. A Processor with a Service Machine Installed

In Figure 81, user FRED is accessing a database machine located on another processor. FRED is located on a processor that does not have a database machine but does have a service machine installed.

National Language Support for Databases

Most error and informational messages are issued from the user machine, but messages and output for operator commands (SHOW and COUNTER), and HELP text are issued from the database machine. Thus if your organization has the database manager on different processors in a collection, you should support the same national languages on the database machine as are supported on the user machines that can access it. For example, if you support French HELP text and messages on a user machine, and a user accesses a database that does not have French HELP text and messages, that user will receive error and informational messages in French, but the HELP text for those messages will be in English. Figure 82 shows an example of such a set-up.

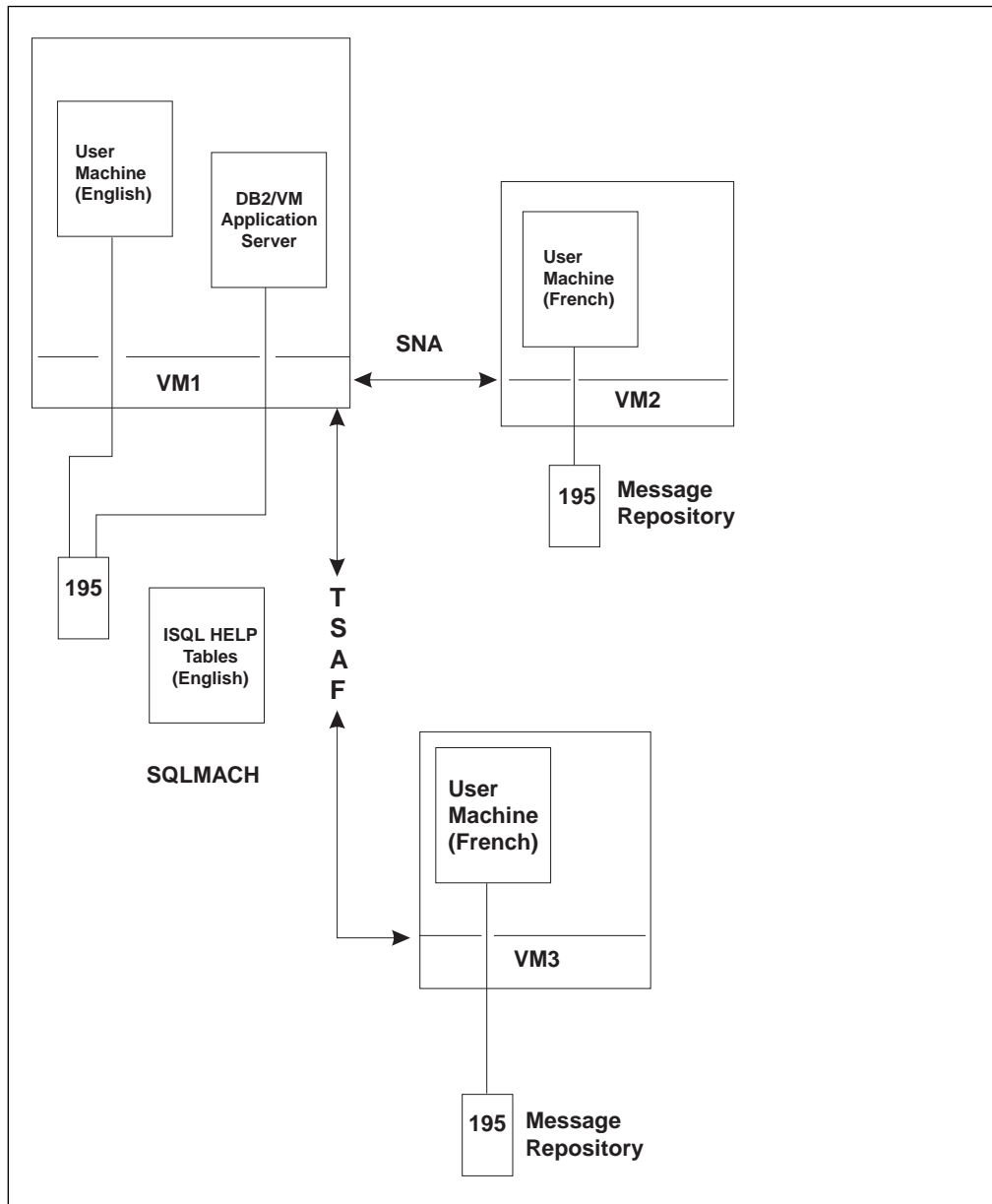


Figure 82. National Language Support for Remote Access Users

Types of Database Machines

When a set-up includes multiple databases, the following terms are used:

- A primary database machine is one that owns a production minidisk: for example, the SQLMACH machine, defined during installation.
- A secondary database machine is one that does not own a production minidisk: for example, the service machine discussed earlier in this chapter. In place of an MDISK control statement for the production minidisk, its VM directory contains the following LINK statement:

```
LINK SQLMACH 195 195 RR
```

which makes the SQLMACH machine's production minidisk available to this secondary database machine. For more information, see "Adding a Secondary Database Machine" on page 292.

- The primary production minidisk is the original production minidisk in your installation. It is owned by the SQLMACH database machine.
- A secondary production minidisk contains a copy of the product-supplied files on the original production minidisk. It is owned by a primary database machine other than the SQLMACH machine.

These definitions are illustrated in Figure 83

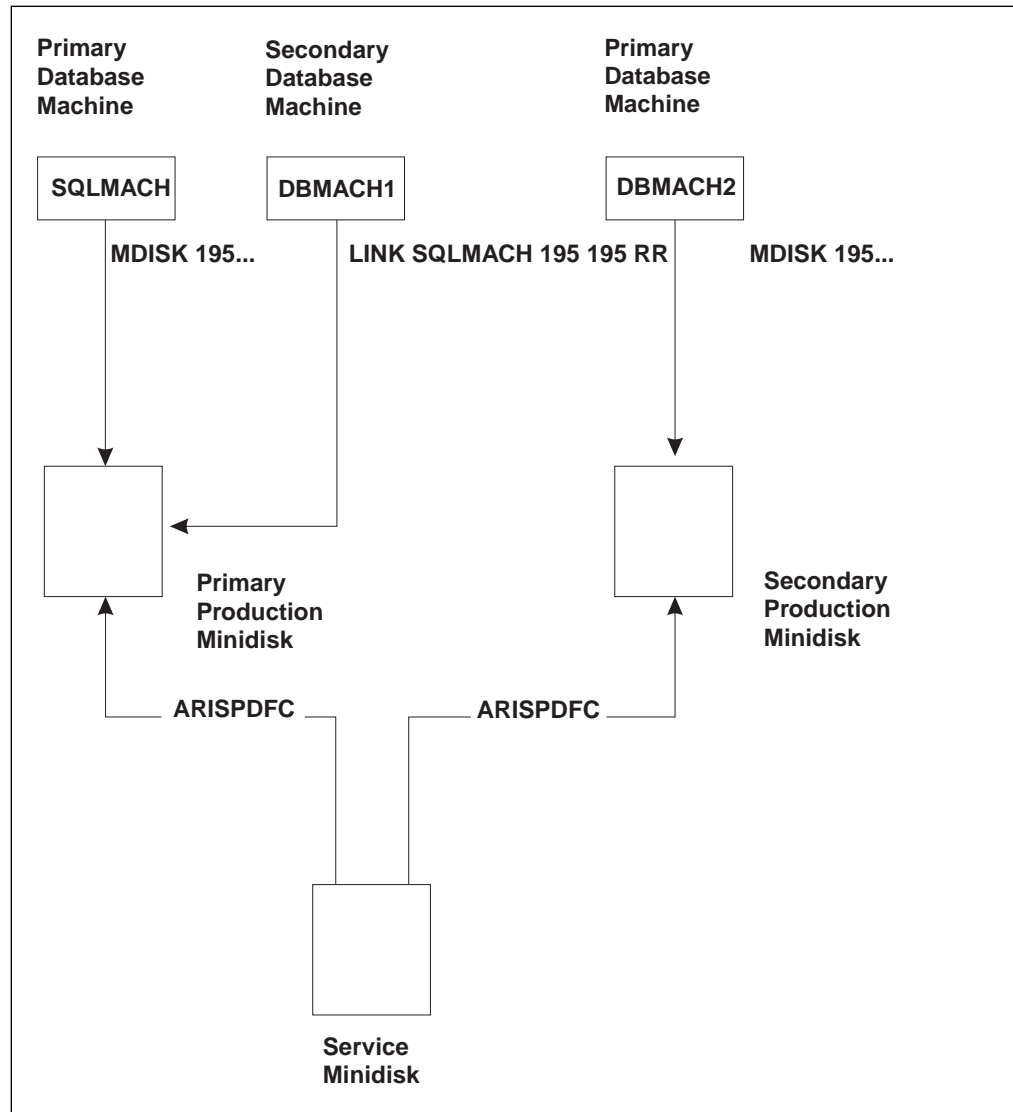


Figure 83. Types of Database Machines

Here, the SQLMACH machine owns the primary production minidisk through an MDISK control statement. The DBMACH1 machine is a secondary database machine, with a LINK directory control statement to the primary production minidisk. The DBMACH2 machine is another primary database machine because it owns a copy of the original production minidisk. This DBMACH2 machine also has a LINK entry for the SQLMACH machine's service minidisk. The service minidisk contains all IBM-supplied files, including those that are necessary for a production minidisk. This link is necessary to prepare the secondary production minidisk by copying all appropriate IBM-supplied files from the service minidisk to it. The ARISPDFC EXEC

is supplied for use in copying these files. For information on using this EXEC, see Appendix G, "Service and Maintenance Execs" on page 493.

The product-supplied files must be the same on all production minidisks at your installation. If you apply service to one production minidisk, you must apply the service to all. (The *DB2 Server for VM Program Directory* describes how to apply service.) When you create a secondary production minidisk, you create a new environment, with its own application server and its own users. You should keep the environments as independent as possible: that is, users should not be allowed to communicate with database machines using different production minidisks. (In the VM/ESA operating system, users can access any production minidisk.)

VM/ESA Operating System Considerations

In VM/ESA operating system, there is no dependency between the database machine's production disk and the database being accessed since the connection by APPC/VM is to a resource identifier rather than a virtual machine. You do not have to copy the virtual machine name to the bootstrap before the resource adapter can communicate with the database.

The resource is identified when SQLSTART is issued. For this reason, resource identifiers must be unique within a TSAF collection or an AVS gateway. The server name (*server_name*) does not have to be the same as the resource identifier (*resid*); however, it should be unique within the SNA network.

In this environment, a user must have a link to a valid production disk. The user can then access any of the application servers in the same collection or network.

Primary Database Machines

Why Add a Database Machine?

The descriptions throughout this manual assume there is only one DB2 Server for VM production minidisk, which is owned by the SQLMACH machine and supports all activity on a VM system. This minidisk contains the SQLFDEF files that identify the databases; the SQLDBN files that identify the database machines and the databases they access; the bootstrap modules that identify where DB2 Server for VM code is to be loaded; and all the code that the database manager needs to run daily. Because it describes the environment, it changes as your environment changes. For example, it is updated if you generate a database, if you add a dbextent to a database, and at times during database machine startup.

In an environment that contains many database machines doing many administrative tasks, having only one production minidisk can lead to contention problems: only one machine at a time can write to the production minidisk, so other machines may have to wait for it to become available. In addition, in environments that contain extremely sensitive data, having only one production minidisk can be a security exposure: every DB2 Server for VM user would have read access to that production minidisk, and could easily determine what databases exist by scanning the files on it.

Thus, while a single production minidisk is suitable for many installations, it does not suit others. For these situations, you can define and use multiple production minidisks.

Adding a Primary Database Machine

One primary database machine already exists in your installation: the SQLMACH machine, which you defined during installation. It owns the original (primary) production minidisk. To define an additional primary database machine, you define and initialize a secondary production minidisk, which is a copy of the original. The new primary database machine owns this new production minidisk through the MDISK statement for this disk in its VM directory entries.

Step 1: Update the VM Directory

Using your local operating procedures, update the VM directory entries for the new primary database machine. Figure 84 shows an example of these statements. The name of the database machine here is *dbmach2*. For a complete description of VM directory control statements, see the *VM/ESA: CP Command and Utility Reference* manual.

```
1 → USER dbmach2 dbmachpw xM xM G
   ACCOUNT nnnnnnnn
2 → OPTION MAXCONN 26
3 → IUCV ALLOW
4 → IUCV *IDENT SQLDBA GLOBAL
5 → IPL CMS
6 → CONSOLE 009 3215 T OPERATOR
   SPOOL 00C 2540 READER *
   SPOOL 00D 2540 PUNCH A
   SPOOL 00E 1403
   LINK MAINT 190 190 RR
7 → LINK MAINT 19D 19D RR
   LINK MAINT 19E 19E RR
8 → LINK SQLMACH 193 193 RR
10 → MDISK 191 3380 cylr 010 volser W
12 → MDISK 195 3380 cylr 024 volser RR readpw writepw multipw
13 → MDISK 200 3380 cylr 034 volxxx R readpw writepw
14 → MDISK 201 3380 cylr 008 volxxx R readpw writepw
15 → MDISK 202 3380 cylr 077 volxxx R readpw writepw
```

Figure 84. Example VM/ESA Directory Control Statements for the Database Machine

Statement 1: USER *dbmach2 dbmachpw xM xM G*

This statement defines the database machine with the VM privilege class G. The recommended minimum virtual size for the SQLMACH machine is 8MB for the base code, or for the base code and DSS. The recommended minimum size is 10MB if the DRDA code is installed.

If you need to perform tape and DASD file load and unload operations for the DBS utility or trace facility, privilege class B is recommended. Privilege class B is needed to attach tapes.

For the virtual machine name and password shown here, replace *dbmach2* and *dbmachpw* with your own machine name and password.

Statement 2: OPTION MAXCONN 26

The MAXCONN value of the VM OPTION control statement determines the maximum number of APPC/VM connections allowed for a virtual machine.

Set this value to be the sum of the following:

- The maximum number of user machines that can be sharing access to the database machine
- The maximum number of database minidisks to be accessed at any given time (directory, logs, and dbextents)
- 1, if you use a VM/ESA operating system, for the connection to the CP identify system service *IDENT
- 1, if you can use TCP/IP communications (that is, the TCPPORT initialization parameter is not zero).

For more information see “Setting the MAXCONN Value” on page 298.

If you plan to use the DB2 Server for VM accounting facility with this database machine, you should also specify the ACCT operand. See Chapter 11, “Using the Accounting Facility” on page 261.

Statement 3: IUCV ALLOW

This statement enables any virtual machine to use APPC/VM or IUCV to initiate communications with the database machine. In order to actually access information in a database, however, a user machine must either have been granted CONNECT authority to the database, or all users must have been granted CONNECT authority (with the ALLUSERS parameter, which is the default). If either of these conditions is met, the user can exercise all privileges granted to PUBLIC.

For more information, see “VM Directory Control Statements” on page 146.

Statement 4: IUCV *IDENT resid LOCAL

Specify the server name (as entered in the DBNAME parameter of the SQLSTART EXEC) and whether the resource is LOCAL or GLOBAL. If LOCAL is specified, only users on the same processor as the application server can access it. If GLOBAL is specified, any user in the same collection as the application server can access it.

In this example, the application server with the resource identifier *resid* is specified as a LOCAL resource.

Note: If remote access is being used, it is recommended you ensure that server names are unique within a set of interconnected SNA networks, and that resids are unique in a TSAF collection or a gateway. (A gateway is also referred to as an LU.) The resid must also be identified with a GLOBAL scope. For more information, see “Distributed Processing Administration” on page 147.

Statement 5: IPL CMS

Uses the CMS saved segment name (for example, CMS) applicable to your VM/ESA environment.

Statement 6: CONSOLE 009 3215 T OPERATOR

If the database machine runs in disconnected mode, you should specify an alternate console. In the statement shown, the console for the OPERATOR virtual machine is identified as the alternate console. For information on running the database machine in disconnected mode,

see “System and DB2 Server for VM Operator Console Considerations” on page 149.

Statement 7: LINK MAINT 19D 19D RR

This is the LINK statement for the CMS Help facility, which is necessary in order for users to access the DB2 Server for VM Help panels for CMS. (Without it, users can still access the HELP text that is stored in a database, as that HELP text is accessed through ISQL, not CMS.)

Statement 8: LINK SQLMACH 193 193 RR

This is the LINK control statement for the service minidisk (the SQLMACH machine’s 193 minidisk). Specify virtual device address 193 and link access mode RR, as shown. (See Figure 86 on page 293.)

All database machines (primary or secondary) should use the same service minidisk, to ensure that they have the same level of service.

Statement 9: LINK SQLMACH 195 195 RR

This is the LINK control statement for the production minidisk (the SQLMACH machine’s 195 minidisk). Specify virtual device address 195 and link access mode RR, as shown. (See Figure 86 on page 293.)

All secondary database machines should share the same minidisk with its primary database machine.

Statement 10: MDISK 191 3380 cylr 010 volser W

This is the MDISK control statement for a read/write work minidisk (A-disk) with virtual device address 191 and link access mode W. The space allocations for various devices are shown in the Installation Requirements and Considerations chapter in the *DB2 Server for VM Program Directory*.

Statement 12: MDISK 195 3380 cylr 020 volser RR *readpw writepw multipw*

This is the MDISK entry for the secondary DB2 Server for VM production minidisk to be owned by this machine. Specify virtual device address 195 and link access mode RR. Both a read password (*readpw*) and write password (*writepw*) must be specified. The space allocations for a production minidisk on various DASD are identified in the Installation Requirements and Considerations chapter in the *DB2 Server for VM Program Directory*.

This secondary production minidisk must be online and available for all DB2 Server for VM operations. When updates to it are required, the product-supplied database generation and maintenance EXECs relink it with access mode M or W.

Specify a multiple-access password (*multipw*) for this secondary production minidisk if:

1. You are defining other database machines to access this machine’s production minidisk.
2. You want one database machine to be able to perform database generation and maintenance activities while others are accessing this minidisk in read mode.

If multiple database machines share a production minidisk, and the database machine that owns the minidisk does not have a multiple-access password (*multipw*) specified in its VM directory MDISK control statement, only the machine that does the database

maintenance can access the production disk with write-access. If database maintenance must be done, the other machines cannot be performing normal processing using read-access to the production minidisk.

Statements 13, 14, and 15: MDISK ... vol xxx R readpw writepw

These are the directory MDISK statements for each minidisk of the starter database. They must have a link access mode R, a read password (*readpw*), and a write password (*writepw*). The same *readpw* and *writepw* should be assigned to all minidisks defined for the database. The minidisk space and virtual disk address requirements for various DASD devices for databases defined using the starter database specifications are identified in the Installation Requirements and Considerations chapter in the *DB2 Server for VM Program Directory*.

Step 2: Prepare the New Database Machine

1. Log on to the database machine.

If this is a new virtual machine, you will receive messages after CMS is loaded, because you do not yet have a PROFILE EXEC or A-disk for the virtual machine. Ignore them and continue.

2. If this is a new virtual machine, format the 191 minidisk (A-disk) by entering the command:

```
FORMAT 191 A
```

Respond to the prompts issued during CMS FORMAT command processing. For a complete description of this command and an explanation of the required prompt responses, see the *VM/ESA: CMS Command Reference* manual.

3. Acquire write access to your 195 minidisk by entering the CP command:

```
LINK dbmach2 195 195 W
```

where *dbmach2* is your database machine ID.

4. Format your secondary production minidisk by entering the CMS command:

```
FORMAT 195 Q
```

Respond to the prompts issued during CMS FORMAT command processing.

5. Create or update the database machine's PROFILE EXEC. The following shows an example of PROFILE EXEC entries.

```
1 —> CP SET RUN ON
2 —> ACCESS 195 Q
3 —> CP TERMINAL LINEND #
4 —> CP TERMINAL LINEDEL OFF
5 —> CP TERMINAL CHARDEL OFF
6 —> CP TERMINAL ESCAPE ¢
7 —> CP SET EMSG ON
8 —> SET LANGUAGE AMENG (ADD ARI USER
```

Figure 85. PROFILE EXEC Entries for an Additional Primary Database Machine

Statement 1: CP SET RUN ON

Required if the database machine will be run in disconnected mode. (Most database machines are run in disconnected mode.)

Statement 2: ACCESS 195 Q

Accesses the secondary production minidisk (virtual device address 195) with file mode Q.

Statements 3, 4, 5, and 6: CP TERMINAL xxxx

Establish the recommended VM TERMINAL line-edit symbols for the database manager.

Statement 7: CP SET EMSG ON

Sets the EMSG function value ON. This is the recommended setting.

Statement 8: SET LANGUAGE AMENG (ADD ARI USER

Ensures a national language message repository is available if there is no CMS saved segment for DB2 Server for VM messages.

6. Run the PROFILE EXEC by entering the command:

```
PROFILE
```

7. Access the service minidisk by entering the CMS command:

```
ACCESS 193 V
```

8. At this time, you should still have access to your secondary production minidisk (195 minidisk, file mode Q). To initialize this minidisk, you must copy the appropriate product-supplied CMS files to it, by running the ARISPDFC EXEC:

```
ARISPDFC
```

For more information on the functions ARISPDFC performs, refer to Appendix G, "Service and Maintenance Execs" on page 493.

VMSES/E Consideration

VMSES/E consideration: ARISPDFC copies over the latest serviced ARISQLLD LOADLIB to the secondary production minidisk. If for some reason you need to create a different version of the ARISQLLD LOADLIB, you must create a PPF (Product Parameter File) override to the DB2 Server for VM \$PPF file (5648A701 \$PPF). Refer to the *VM/ESA: VMSES/E Introduction and Reference* for information on creating PPF overrides. In your new PPF override, you must:

- Reflect a new local modification disk, which has the different versions of the TEXT files to be included into the ARISQLLD LOADLIB as local modifications.
- Change the installation user ID and test production address or SFS directory name to reflect the secondary user ID and production minidisk or SFS directory.
- Change the PPF build section to bypass all builds except for the one for the LOADLIB.

End of VMSES/E Consideration

9. Next, acquire read access to your production minidisk by entering:

```
LINK dbmach2 195 195 RR
```

```
ACCESS 195 Q
```

where *dbmach2* is your database machine ID.

Note: Do not log off yet.

Step 3: Generate a Database

The process of defining an additional primary database machine also generates a database. In this example, the specifications for the starter database are used. If you want to generate a different database, review Chapter 2, “Planning for Database Generation” on page 15.

To generate a starter database, enter:

```
SQLDBINS DBNAME(name) STARTER(YES) RESID(resid)
```

where *name* is the name of the database to be generated.

Note: If DBNAME is not the same as RESID, you must specify the *resid* option.

The following prompt appears:

```
ARI6010D Do you want to install English DB2 Server for VM HELP text?  
Enter 0(No), 1(Yes), or 111(Quit).
```

If you need information on how to respond to this prompt, see “Running the SQLDBINS EXEC” on page 299.

After the database is generated, you can log off the new primary database machine.

For information on installing HELP text in additional languages, see “National Language Support for Messages and HELP Text” on page 348.

Adding a Secondary Database Machine

This section describes how to define an additional database machine. Because a database machine usually owns one or more databases, this section also shows how to generate a database for the new machine.

Before proceeding, you must know the read and write (or multiple access) passwords for the DB2 Server for VM production minidisk.

Step 1: Update the VM Directory

Using your local operating procedures, update the VM directory entries for the new database machine. Figure 86 shows an example. The name of the database machine is *dbmach1*. These statements are the same as those for a primary database machine: for an explanation of them, see “Step 1: Update the VM Directory” on page 287. For a complete description of these statements, see the *VM/ESA: CP Command and Utility Reference* manual.

```

1 → USER dbmach1 dbmachpw xM xM G
   ACCOUNT nnnnnnnn
2 → OPTION MAXCONN 25
3 → IUCV *IDENT resid LOCAL
4 → IUCV ALLOW
5 → IPL CMS PARM AUTOOCR
6 → CONSOLE 009 3215 T OPERATOR
   SPOOL 00C 2540 *
   SPOOL 00D 2540 A
   SPOOL 00E 1403
   LINK MAINT 190 190 RR
7 → LINK MAINT 19D 19D RR
   LINK MAINT 19E 19E RR

8 → LINK SQLMACH 193 193 RR
9 → LINK SQLMACH 195 195 RR
10 → MDISK 191 3380 cylr 010 volser W
13 → MDISK 200 3380 cylr 034 volxxx R readpw writepw
14 → MDISK 201 3380 cylr 008 volxxx R readpw writepw
15 → MDISK 202 3380 cylr 077 volxxx R readpw writepw

```

Figure 86. VM Directory Control Statements for a Secondary Database Machine

Use the CMS saved segment name (for example CMS, CMSL) and device types applicable to your VM operating system.

Step 2: Prepare the Database Machine

1. Log on to the database machine and IPL CMS.

If this is a new virtual machine, you will receive messages after CMS is loaded because you do not yet have a PROFILE EXEC or A-disk for the virtual machine. Ignore them and continue.

2. If this is a new virtual machine, format the 191 minidisk (A-disk) by entering the command:

```
FORMAT 191 A
```

Respond to the prompts issued during CMS FORMAT command processing. For a complete description of CMS commands and an explanation of the required prompt responses, see the *VM/ESA: CMS Command Reference* manual.

3. Create or update the database machine's PROFILE EXEC. Figure 87 shows an example. These statements are the same as those for a primary database machine: for an explanation of them, see Figure 85 on page 290.

Note that the entries are provided in file SQLMACH PROFILE located on the production disk.

4. Run the PROFILE EXEC by entering the command:

```
PROFILE
```

Note: Do not log off yet.

1	→	CP SET RUN ON
2	→	ACCESS 195 Q
3	→	CP TERMINAL LINEND #
4	→	CP TERMINAL LINEDEL OFF
5	→	CP TERMINAL CHARDEL OFF
6	→	CP TERMINAL ESCAPE ¢
7	→	CP SET EMSG ON
8	→	SET LANGUAGE AMENG (ADD ARI USER

Figure 87. PROFILE EXEC Entries for an Additional Database Machine

Step 3: Generate a Database

To generate a database, you must first establish access to the service minidisk. Enter the following command:

```
ACCESS 193 V
```

In this example, you will generate a database using the starter database specifications. Before proceeding, review the minidisk requirements shown in *DB2 Server for VM Program Directory*. If you choose to generate your own database, review Chapter 2, “Planning for Database Generation” on page 15.

To generate a starter database enter:

```
SQLDBINS DBNAME(name) STARTER(YES) RESID(name)
```

where *name* is the name of the database to be generated.

Note: If DBNAME is not the same as RESID, you must specify the *resid* option.

The following prompt appears:

```
ARI6010D Do you want to install English SQL/DS HELP text?
Enter 0(No), 1(Yes), or 111(Quit).
```

If you need information on how to respond to this prompt, see “Running the SQLDBINS EXEC” on page 299.

After the database is generated, you can log off the new database machine.

For information on installing HELP text in additional languages, see “National Language Support for Messages and HELP Text” on page 348.

Adding a Service Machine

Use VMSES/E to install the database manager code on that processor, but do not run the database installation utilities. See Chapter 1, “Planning for Installation” on page 1 and the *DB2 Server for VM Program Directory*.

Defining Additional User Machines

To define additional user machines you can either update existing virtual machines or define new ones. In either situation, you must update the VM directory. The new user machine may also require a CMS communications directory. For more information, see “Setting Up the CMS Communications Directory” on page 12.

You can update the VM directory using your installation's current operating procedures. For a complete description of VM directory control statements, see the *VM/ESA: CP Command and Utility Reference* manual for your IBM VM system.

Figure 88 shows an example of the VM directory entries for a user machine.

```

1  —> USER sqluser userpw xM xM G
2  —> ACCOUNT nnnnnnnn
3  —> IUCV resid
      IPL CMS PARM AUTOCR
      CONSOLE 009 3215
      SPOOL 00C 2540 *
      SPOOL 00D 2540 A
      SPOOL 00E 1403
      LINK MAINT 190 190 RR
4  —> LINK MAINT 19D 19D RR
5  —> MDISK 191 3380 cylr 010 volser W
6  —> LINK SQLMACH 195 195 RR

```

Figure 88. VM Directory Entries for a User Machine

Use the CMS saved segment name (for example, CMS, CMSL) and device types applicable to your VM environment.

Statement 1: USER *sqluser userpw xM xM G*

This statement defines the user machine with the VM privilege class G. The virtual size required for the user machine is 6 megabytes .

Privilege class B is optional. It is only required by a user machine that needs to attach real devices, such as tape drives.

Statement 2: ACCOUNT *nnnnnnnn*

This statement supplies an account number for the user machine.

Statement 3: IUCV *resid*

This statement is optional. It enables the user machine to connect to the specified database (*resid*). If you want a user machine to communicate with more than one database, specify more than one IUCV *resid* statement.

Alternatively, user machines can communicate with databases without the need of the IUCV *resid* statement if both the following are true:

- The database is located on the same processor, or is defined as a GLOBAL resource on another processor (within the same collection).
- The database machine that owns the database has the IUCV ALLOW statement defined in its VM directory.

For more information on the VM directory control statements that affect inter-machine communications, see “VM Directory Control Statements” on page 146.

Statement 4: LINK MAINT 19D 19D RR

This is the LINK statement for the CMS Help facility, which is necessary in order for users to access the DB2 Server for VM Help panels for CMS. (Without it, users can still access the HELP text that is stored in a database, as that HELP text is accessed through ISQL, not CMS.)

Statement 5: MDISK 191 3380 cylr 003 volser W

This is the MDISK control statement for a read/write work disk (A-disk) with virtual device address 191 and link access mode W specified. The 191 minidisk space allocations for various DASD devices are shown in the Installation Requirements and Considerations chapter in the *DB2 Server for VM Program Directory*.

Statement 6: LINK SQLMACH 195 195 RR

This is the LINK control statement for a production minidisk with virtual device address 195 and link access mode RR specified; it establishes the virtual machine as a user machine. (All user machines must contain this statement.)

Initialize the user machine.

1. Log on to the user machine.

If this is a new virtual machine, you will receive messages after CMS is loaded, because you do not yet have a PROFILE EXEC or A-disk for the virtual machine. Ignore them and continue.

2. If this is a new virtual machine, format the 191 minidisk (A-disk) by entering the command:

```
FORMAT 191 A
```

Respond to the prompts issued during CMS FORMAT command processing. For a complete description of the FORMAT command and an explanation of the required prompt responses, see the *VM/ESA: CMS Command Reference* manual.

3. Create or update the user machine's PROFILE EXEC. Figure 89 shows an example.

Note that the entries are provided in the file SQLUSER PROFILE located on the production disk.

4. Log off the user machine.

<pre>1 → ACCESS 195 Q 2 → CP SET EMSG ON 3 → SET LANGUAGE AMENG (ADD ARI USER</pre>

Figure 89. PROFILE EXEC Entries for a User Machine

Statement 1: ACCESS 195 Q

Accesses the production minidisk (virtual device address 195) with file mode Q.

Statement 2: CP SET EMSG ON

Sets the VM EMSG function value ON. This is the recommended setting.

Statement 3: SET LANGUAGE AMENG (ADD ARI USER

Ensures a national language message repository is available if there is no CMS saved segment for DB2 Server for VM messages.

With the user machine defined this way, users can issue the SQLINIT EXEC to communicate with a database machine, and set up a default database for the user.

If you have users who typically access the same database all the time, who do not use the database manager frequently, or who are inexperienced in data processing, you may want to issue the SQLINIT EXEC while doing the above steps to set up their machines for accessing the appropriate database.

Adding a Database

You can add an additional database to any of the database machines that you have defined. Before doing so, review Chapter 2, “Planning for Database Generation” on page 15.

Database generation consists of a series of steps that define and format minidisks, initialize the database, preprocess the DBS utility package, and run the DBS utility to complete the basic generation process. After the database is defined, other DB2 Server for VM facilities are installed in it. Most of these subtasks are done by issuing a single IBM-supplied EXEC called SQLDBINS, which calls other IBM-supplied EXECs.

These steps are discussed in detail below.

Step 1: Defining the Database Minidisks

The first step is to add MDISK control statements to the VM directory of a database machine to give that machine ownership of the database you are about to generate.

Initially, the only database machine on a VM system is SQLMACH, but you can define additional machines as described in “Adding a Primary Database Machine” on page 287. Each database machine can have more than one database (although it can only access one at a time), and can access databases that it does not own. For convenience, the owner should be the machine that will be the primary user or controller.

The specific MDISKS required depend on the requirements for your database. However, you **must** define VM minidisks for:

- A directory (sometimes referred to as a BDISK)
- Either one or two database logs (one log is mandatory and two logs are required to support dual logging)
- At least one database dbextent.

Figure 90 shows an example of the MDISK control statements needed to define minidisks for a database that has two logs and two dbextents. For detailed information on the MDISK control statement, see the *VM/ESA: CP Command and Utility Reference* manual.

```
MDISK 300 3380 cylr 6   XXXXXX R DBPSW XLMT
MDISK 324 3380 cylr 32  XXXXXX R DBPSW XLMT
MDISK 32A 3380 cylr 523 YYYYYY R DBPSW XLMT
MDISK 32B 3380 cylr 516 XXXXXX R DBPSW XLMT
MDISK 32D 3380 cylr 32  YYYYYY R DBPSW XLMT
```

Figure 90. Defining VM Minidisks for a Database

Notes:

1. The first statement defines a minidisk for the database directory, and gives it a virtual device address of 300.
2. The next statement defines the first log minidisk for the database, and gives it a virtual device address of 324.
3. The next two statements define two dbextent minidisks, one for each storage pool to be used initially, and give them virtual device addresses of 32A and 32B.
4. The last statement defines the optional second log minidisk and gives it a virtual device address of 32D.
5. Keep a record of these virtual device addresses since you will later need to supply them to the EXEC that generates the database.
6. For minimum space allocation values, see Table 43 on page 452.
7. This example assumes a count-key-data device. Thus, you must supply a cylinder relocation factor (*cylr*). For an FB-512 device, specify the appropriate block relocation factor (*blkr*) and number of blocks (*blks*).
8. The volume serial number represented by *XXXXXX* contains the directory, one log, and one dbextent. The volume serial number represented by *YYYYYY* contains one log and one dbextent. Set these volume serial numbers as appropriate for your DASD volumes.
9. All the MDISK control statements must have an access mode of R.
10. For DBPSW and XLMT, specify your own read and write passwords and give them an access mode of R.
11. Do not add LINK control statements to the VM directory for database minidisks. Database machines will be linked to these minidisks when the database manager is started.

The database manager does not place any restrictions on the virtual device addresses you can use for its minidisks, unless you are using the starter database specifications, in which case the minidisks will occupy virtual device addresses 200, 201, and 202.

Setting the MAXCONN Value: When adding MDISK control statements for a database machine, you should also review the current setting for MAXCONN, which is a parameter of the VM OPTION control statement. The MAXCONN determines the number of APPC/VM connections allowed for a virtual machine, and is unique to each database machine. Set it to be the sum of:

1. The maximum number of user machines that can share access to the database machine at the same time

Each user machine that communicates with the database machine requires an APPC/VM or IUCV connection.

2. The maximum number of database minidisks -- the directory, the log or logs, and the dbextents -- to be accessed at any given time.

The database manager uses DASD block I/O, which in turn uses IUCV, to access its database minidisks. Set the MAXCONN value to allow for the maximum number of minidisks that are attached to the database machine.

Because the database manager records the highest dbextent number for the database, you may have to count deleted dbextents when calculating the MAXCONN value. For example:

- If you initially have 10 dbextents and you delete dbextent number 2, you still count 10 dbextents.
 - If you initially have 10 dbextents, delete dbextent number 2, and then add another dbextent number 2 to replace the deleted one, count 10 dbextents.
 - If you initially have 10 dbextents, delete dbextent number 3, and then add dbextent number 11, count 11 dbextents.
3. The value of MAXCONN must be increased by 1 to allow for the connection to the CP identify system service *IDENT.
 4. The value of MAXCONN must be increased by 1 if you are using TCP/IP communications with the server (that is, if you have set the TCPSPORT initialization parameter to a non-zero value).

For example, suppose there are 50 virtual machines (users) allocated for the installation, including the database machine SQLMACH. Of the 49 user machines, 25 are communicating with SQLMACH. In addition, assume that SQLMACH owns three databases: one with 10 minidisks, one with 3 minidisks, and one with 15 minidisks. For the SQLMACH machine, MAXCONN should be set to 40, that is, 25 (for maximum number of connected machines) + 15 (for the largest number of database minidisks). Note that this second number is **not** the sum of the minidisks in all three databases, because the database machine can only run one database at a time. If SQLMACH were to access the database having 3 minidisks, 37 user machines could connect. This (in most cases) is better than setting MAXCONN to 28 (to accommodate the small database and 25 users), because then if the largest database is accessed, only 13 user machines would be able to connect.

Naturally, you may have to adjust MAXCONN as real conditions (number of minidisks and number of users) change.

The MAXCONN parameter also applies to single user mode; however, here it is only concerned with the number of minidisks because no other user machines are communicating with the database machine.

Step 2: Generating a Database

To generate a database:

1. Log on to the database virtual machine that owns the minidisks
2. Get read access to the service minidisk (ACCESS 193 V)
3. Run the SQLDBINS EXEC

This third step is described in detail below.

Running the SQLDBINS EXEC: The SQLDBINS EXEC resides on the service minidisk, where it is placed during the installation process. Figure 91 on page 300 shows its format.

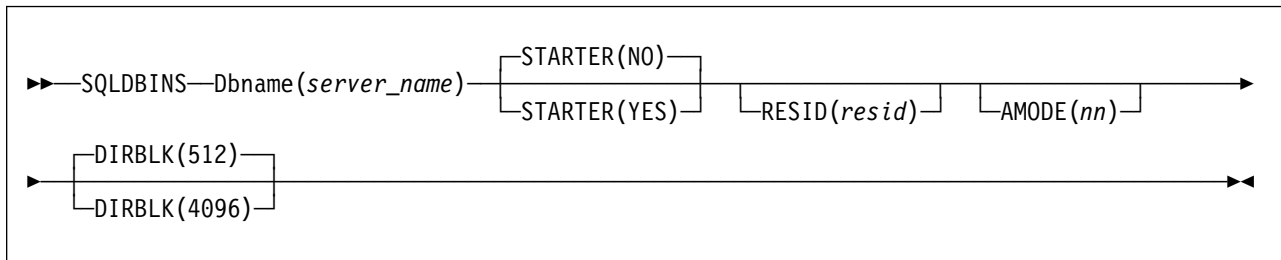


Figure 91. SQLDBINS EXEC

DBNAME

The DBNAME parameter must be specified. For *server_name*, specify a name for the database. If the IUCV *IDENT directory entry of the database machine specifies a resource name, then the *server_name* specified when SQLDBINS is invoked must match the resource name. If the IUCV *IDENT directory entry specifies RESANY, then any valid dbname can be specified. The database name you use must match the name defined in the IUCV *IDENT directory entry of the database machine. Because the production minidisk defines a complete environment, the name must be unique among all those database names already existing on the production minidisk. Also, the name must not be greater than 18 characters.

STARTER

If you want to generate a database using the starter database specifications, specify STARTER(YES). The starter database will be defined on virtual device addresses 200, 201, and 202, and you will not be able to change these addresses. (Thus, you can generate only one starter database for each database machine.) STARTER(NO) is the default. For more information on the starter database specifications, see Chapter 2, “Planning for Database Generation” on page 15.

RESID

If *server_name* is greater than 8 characters, you must also specify a *resid* of up to 8 characters; if *server_name* is 8 characters or fewer, *resid* defaults to *server_name*, unless you specify another *resid*.

For *resid*, specify a VM resource identifier for the database that is unique on the production minidisk being accessed.

AMODE

The AMODE(24) option only needs to be specified when user-written exits do not support 31-bit addressing. In this case, AMODE(24) must be specified when running the SQLDBINS EXEC. For a description of this parameter, see “AMODE” on page 60. For more information on virtual machine operating modes, see “Running the Database Manager” on page 95.

DIRBLK

Specifies whether the SQLDBGEN EXEC will generate the new directory in 512 or 4KB blocks. The default is 512. This parameter is only effective if you answer YES to message ARI0647D, which asks whether you want to format the directory disk. DIRBLK(4096) is only valid if you have installed the Data Spaces Support code.

The SQLDBINS EXEC installs a database complete with DBS utility support, ISQL support, and English HELP text. If you do not use these facilities, you should leave the support in the database anyway.

If, for any reason, SQLDBINS ends before its processing is complete (for example, system failure or user interrupt), rerun it.

The SQLDBINS EXEC does much of its work by calling other IBM-supplied EXECs, most of which run without your intervention. One of these is the SQLDBGEN EXEC, which does the actual database generation. Thus, before issuing SQLDBINS, you must prepare the proper input for SQLDBGEN as follows. (If you are using the starter database specifications, you do not have to prepare any input.)

Preparing Input for the SQLDBGEN EXEC: The SQLDBGEN EXEC resides on the service minidisk, where it is placed during the installation process. Figure 92 shows its format.

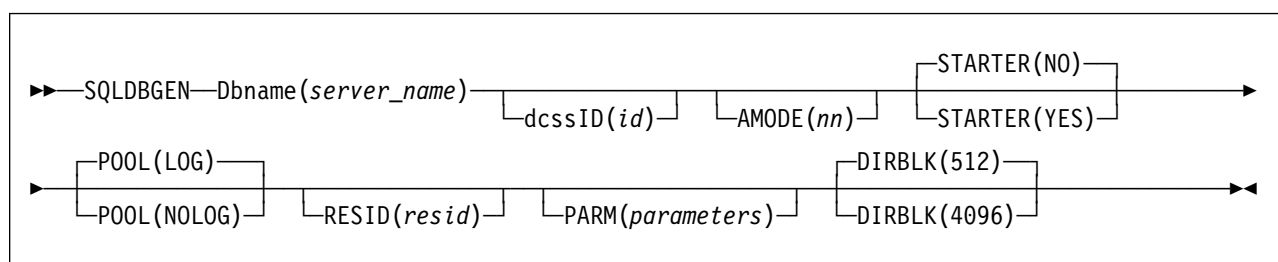


Figure 92. SQLDBGEN EXEC

When the SQLDBGEN EXEC is called by the SQLDBINS EXEC, some of these parameters are omitted, while others take on the value that is specified on SQLDBINS. You only need to complete all of them if you are calling the SQLDBGEN EXEC directly, which you might do if you are moving databases from a VSE to a VM system, or doing problem diagnosis. For more information on moving a database from a VSE system to a VM system, see “Migrating from a VSE to a VM Operating System” on page 42.

DBNAME

The DBNAME parameter must be specified; you can use either DBNAME or any valid initial substring (that is, you can use D, DB, DBN, DBNA, or DBNAM). For *server_name*, specify a name for the application server. The name must be unique among all other database names defined for the production minidisk being accessed, and not more than 18 characters in length. If it is more than 8 characters, you must also specify RESID.

When SQLDBINS calls SQLDBGEN, the DBNAME specified on SQLDBINS is passed to SQLDBGEN.

DCSSID

This parameter is optional. You should specify this parameter only if you have created saved segments for the DB2 Server for VM code and want to use those saved segments. DCSSID specifies where the code is to be loaded in this virtual machine. You can specify ID instead of DCSSID for the keyword: no other abbreviation is valid. For more information on loading DB2 Server for VM code into different areas of storage, refer to “Defining Saved Segments” on page 187.

The SQLDBINS EXEC operates on the assumption that no saved segment has yet been defined; thus, when calling SQLDBGEN, it omits the DCSSID parameter.

AMODE

This parameter is optional and specifies the addressing mode the database manager uses. For a description of this parameter, see “AMODE” on page 60.

The AMODE, if any, specified on SQLDBINS, is passed to SQLDBGEN.

STARTER

The SQLDBINS EXEC uses the STARTER parameter. Omit it when running SQLDBGEN directly. STARTER(NO) is the default.

POOL

This parameter is optional. Specify POOL(LOG) to indicate that you want to define storage pools for which the database manager does full recovery. Specify POOL(NOLOG) to indicate that you want to have the opportunity to define storage pools that are nonrecoverable. For more information, see “Nonrecoverable Storage Pools” on page 250.

If you specify POOL(NOLOG), SQLDBGEN prompts you for the numbers of any nonrecoverable storage pools you want to create. If you omit the POOL parameter or specify POOL(LOG), you will not receive any prompts.

The SQLDBINS EXEC always omits this parameter when calling SQLDBGEN.

RESID

You must specify RESID if DBNAME is greater than 8 characters. However, if DBNAME is less than 8 characters, RESID does not need to be specified because in this case, DBNAME and RESID may be the same.

For RESID, specify a VM resource identifier for the database. This identifier must be unique on the production minidisk being accessed.

The RESID, if any, specified on SQLDBINS, is passed to SQLDBGEN.

PARM

Normally, you should omit this parameter: only use it to specify additional parameters for problem determination - for example, DUMPTYPE or TRACRDS. It can only be specified when invoking SQLDBGEN directly, and must always be last. For detailed information on how to use the PARM parameter, see Chapter 4, Planning for Operation of the Database Manager.

Note: The SQLDBGEN EXEC automatically specifies SYSMODE=S, STARTUP=C, and DUALLOG (Y or N, as appropriate) when it calls SQLSTART, so do not specify those parameters for PARM. Also, do not specify the PROGNAME initialization parameter as it is not valid when STARTUP=C.

DIRBLK

Specifies whether the SQLDBGEN EXEC will generate the new directory in 512 or 4KB blocks. The default is 512. This parameter is only effective if you answer YES to message ARI0647D, which asks whether you want to format the directory disk. DIRBLK(4096) is only valid if you have installed the Data Spaces Support code.

The SQLDBGEN EXEC does three major tasks:

1. Formats and reserves the database minidisks.
2. Creates a file called *resid* SQLFDEF that contains the CMS FILEDEF and CP LINK commands for the minidisks.

3. Starts the database manager in single user mode with the coldstart option (STARTUP=C) to format the database components, generate the catalog tables, and create a CMS file called *resid* SQLDBN.

These three operations are discussed below.

Format and Reserve the Database Minidisks: For each minidisk in the database, SQLDBGEN issues a CMS FORMAT and a CMS RESERVE command. It prompts you for the virtual device address (*cuu*) of each minidisk, which you should have recorded when you specified the MDISK statements in “Step 1: Defining the Database Minidisks” on page 297. Because the FORMAT and RESERVE commands will erase the current contents of the minidisk, you will be asked to confirm that you really want to have them run: respond YES.

Attention: Be sure you are accessing the correct minidisk before you do so.

The directory is formatted with a block size of 512 bytes. All other minidisks are formatted with a block size of 4096 bytes.

Note: When using DB2 Data Spaces Support, the directory could be formatted with a block size of 4096 bytes.

The RESERVE command creates CMS-like database files on the minidisks. These files are not the same as regular CMS files.

Note: Never use CMS file manipulation commands (such as COPY or RENAME) against the database files, as these commands may render the database useless. Instead, use DB2 Server for VM facilities. For a list of commands that you should not use, see “CMS Restrictions” on page 148.

Create SQLFDEF: Each database must have one (and only one) SQLFDEF CMS file. The SQLDBGEN EXEC creates this file and places entries in it after it issues FORMAT and RESERVE for each minidisk. All SQLFDEF files reside on the production minidisk (Q-disk).

Note: The SQLDBGEN EXEC accesses the production minidisk in write access mode M, which allows a database machine to get write access to the production minidisk while someone else has read access. If another database machine already has write access, both SQLDBGEN and SQLDBINS end, and you must restart SQLDBINS when no other machine has write access.

The CMS file name of the SQLFDEF file is taken from the DBNAME parameter you specified when you ran SQLDBINS. For example, if you specified SQLDBINS DB(DBNAME01), a file called DBNAME01 SQLFDEF is created on the production minidisk. You can determine whether a database exists by issuing the CMS STATE command for the SQLFDEF file. For example, to determine whether there is a database called PROD34, issue:

```
STATE PROD34 SQLFDEF Q
```

Before creating the SQLFDEF file and before formatting the database minidisks, SQLDBGEN issues a STATE command similar to the one above to determine whether the database specified in the DBNAME parameter (of SQLDBINS) already exists. If it does, SQLDBGEN gives you the choice of either using the existing file or replacing it. If you choose to use it, you are given a choice of whether to have SQLDBGEN issue FORMAT and RESERVE commands for the minidisks.

The SQLFDEF file contains CP LINK and CMS FILEDEF commands for the database minidisks. When the SQLSTART EXEC is called to start the database manager, it uses the SQLFDEF file as the vehicle by which it accesses the database. Specifically, it copies the SQLFDEF file from the production minidisk to the database machine's A-disk, and changes the file name and file type to ARISFDEF EXEC (any existing CMS file with this name); then calls ARISFDEF EXEC to access the minidisks of a particular database.

The minidisk read and write passwords do not appear in the LINK commands in SQLFDEF: the only place they appear is in the VM directory entry for the database machine that owns the database. This prevents users who do not know the passwords from accessing the database minidisks. For more information on using the SQLFDEF file to maintain security, and for information on how to protect the minidisks, see Chapter 6, "Maintaining Database Security" on page 143.

The FILEDEF commands in SQLFDEF associate the virtual device addresses of the database minidisks with the ddnames used by the DB2 Server for VM program. The database manager uses the following stylized ddnames for its directory, log, and dbextent minidisks:

- Directory minidisk: BDISK
- Log minidisk (or minidisks): LOGDSK1 (and, optionally LOGDSK2)
- Dbextent minidisk (or minidisks): DDSK1 (through DDSK n).

Because the FILEDEFs for the program are generated for you, these ddnames are not usually a concern. Many people refer to the minidisks by their ddnames, so it is good to keep the ddnames in mind. The ddnames can also appear in messages to identify a minidisk.

Do a Coldstart: After FORMAT and RESERVE are issued for the minidisks and SQLFDEF is created, the VM mechanisms for using the database minidisks are in place. Next, the database components must be formatted and the catalog tables created. The SQLDBGEN EXEC does these tasks by doing a coldstart: that is, starting the database manager in single user mode with STARTUP=C. To do this, it calls another IBM-supplied EXEC: SQLSTART. As part of its processing, the SQLSTART EXEC either creates or updates a CMS file named *resid* SQLDBN. The *resid* is the name that identifies the database to the VM system.

The SQLDBN file is created on the production minidisk. It contains four pieces of information:

1. The name of the database machine that called SQLSTART.
2. The name of the database being accessed (the server name specified in the DBNAME parameter).
3. The *dcssid* optionally specified in the DCSSID parameter. The default DCSSID is SQLDBA.
4. The AMODE in which the database manager will run.

The SQLDBN file is used to establish communications between users and database machines. Its uses are discussed in Chapter 4, Planning for Operation of the Database Manager; at this point, all you need to know is that it exists, and that you should not erase it after database generation. Like the SQLFDEF file, this file exists on the production minidisk for the life of the database.

Of more concern at this point is the coldstart processing, which requires you to provide input about the remaining database parameters in the form of database generation control statements. To help you provide this input, the SQLDBGEN EXEC first searches for a file named *resid* SQLDBGEN on the production disk. If this file exists, it means that a database with the resource identifier *resid* has already been generated, so you are prompted to either use this file or replace it. If you choose to use it, SQLDBGEN copies the file to the database machine's A-disk by issuing a CMS COPYFILE command similar to this one:

```
COPYFILE resid SQLDBGEN Q resid SQLDBGEN A (REP
```

Any existing file with the name *resid* SQLDBGEN will be replaced.

If you choose not to use this file, or if it cannot be found, SQLDBGEN copies a file named ARISDBG MACRO from the service minidisk to the database machine's A-disk, by issuing a CMS COPYFILE command similar to this one:

```
COPYFILE ARISDBG MACRO V resid SQLDBGEN A (REP
```

Any existing *resid* SQLDBGEN file on your A-disk will be replaced.

The ARISDBG MACRO contains the control statements that are used in generating the starter database. Usually the file that is copied is ARISDBG MACRO, not the production minidisk *resid* SQLDBGEN file. (That is, usually you generate a new database rather than replace an existing one.)

Figure 93 shows the ARISDBG MACRO.

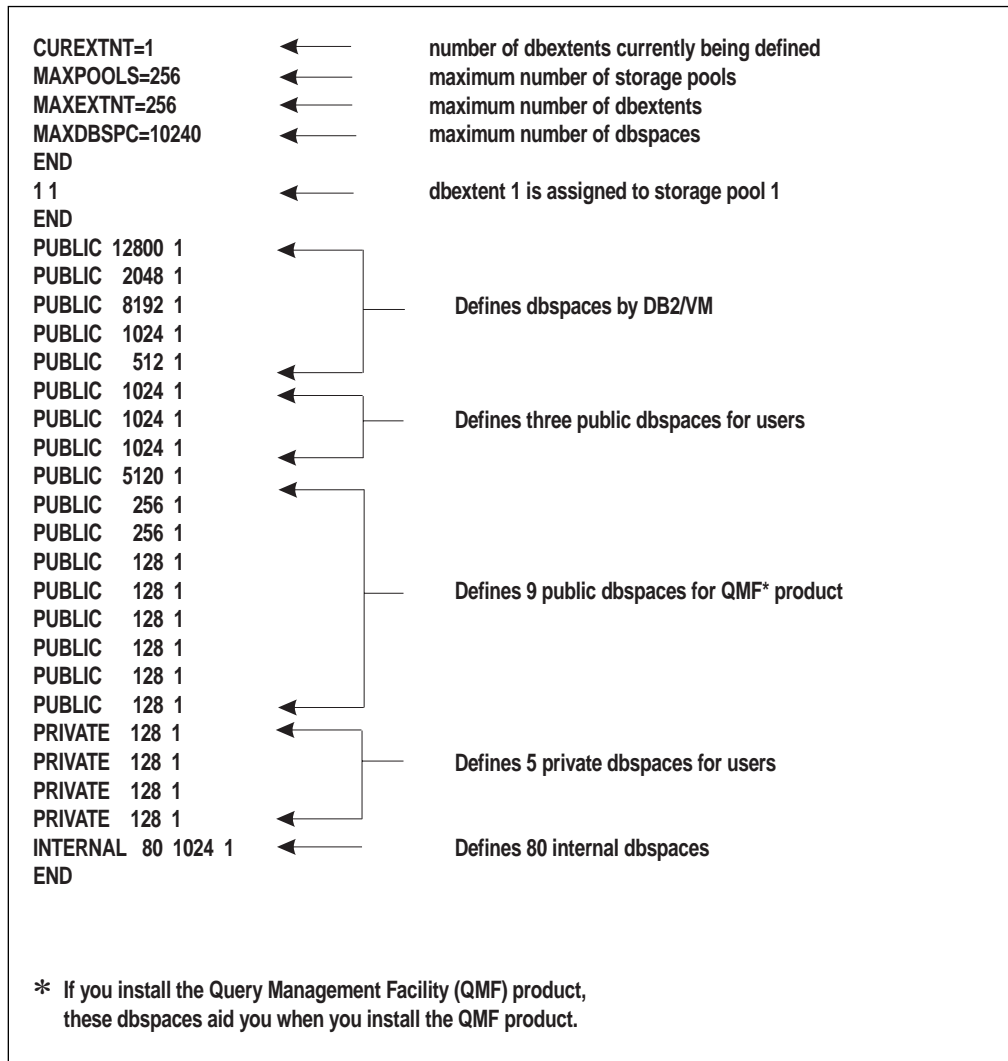


Figure 93. Database Generation Control statements for a Starter Database

If you specify POOL(NOLOG) when using SQLDBGEN, you are prompted for the numbers of storage pools that you want to be nonrecoverable.

Remember that the SQLDBINS EXEC calls the SQLDBGEN EXEC without the POOL parameter. Hence, during a normal database generation, you will not be prompted for information on nonrecoverable storage pools. If you want to define such storage pools during database generation, enter the POOL keyword control statements when you are given the opportunity to edit the *resid* SQLDBGEN file.

The SQLDBGEN EXEC generates additional database generation control statements based on the numbers you provide. The control statement generated is the POOL keyword control statement. (The POOL keyword control statement, along with all the other database generation keyword control statements, is described later in this section.)

After the SQLDBGEN EXEC has generated the POOL keyword control statements (if any), it asks if you want to modify the file. (It does this regardless of what you have specified for its POOL parameter.) If you answer yes, SQLDBGEN starts XEDIT.

You should modify the CMS file as appropriate and FILE it. (Note that the comments on the right of the keyword control statements are not in the actual file.) The SQLDBGEN EXEC then copies the *resid* SQLDBGEN file to the production minidisk (this provides a record of the control statements used to generate a given database), and then does a coldstart of the database manager. The database manager uses the completed file to format the database components and to create the catalog tables. The SQLDBGEN EXEC then ends.

Figure 94 shows how you might modify the CMS file for a database. The control statements shown work with the minidisk definitions in Figure 90 on page 297. Table 2 on page 15 summarizes all database generation parameters.

```

CUREXTNT=2
MAXEXTNT=200
MAXDBSPC=1000
END
POOL 2 NOLOG
1 1
2 2
END
PUBLIC    12800 1
PUBLIC    2048 1
PUBLIC    8192 1
PUBLIC    1024 1
PUBLIC    512 1
PUBLIC    512 1
PUBLIC    512 2
PUBLIC    512 2
PRIVATE   128 1
PRIVATE   128 1
PRIVATE   512 1
PRIVATE   128 2
PRIVATE   128 2
PRIVATE   512 2
INTERNAL  80 1024 1
END

```

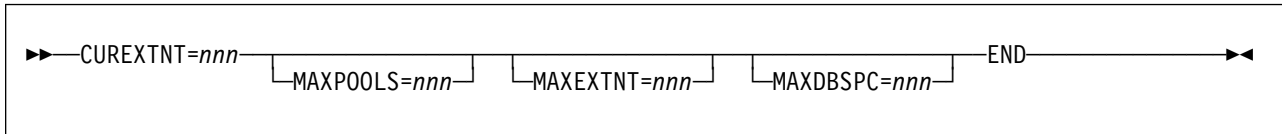
Figure 94. Example Database Generation Control Statements

The three control statements are divided into sets of input records, called keyword control statements, and are separated by END delimiter control statements. If you specify more than one keyword control statement on a single input record, separate the statements by a blank. The END delimiter control statement cannot be combined with the other keyword control statements. The keyword control statements must be coded in columns 1-72.

The details of specifying these database generation control statements are described below.

Specifying Database Maximum Values: These keyword control statements define the number of dbextents to be initialized during the database generation process (CUREXTNT), and establish certain maximum values for the database (MAXPOOLS, MAXEXTNT, and MAXDBSPC).

The format for specifying these values is :



CUREXTNT

CUREXTNT specifies the number of dbextents being defined in the database generation. You must specify CUREXTNT; there is no default value. Its value can be from 1 to 999, and it must match the number of dbextent definition keyword control statements. In the example shown in Figure 94 on page 307, CUREXTNT=2, indicating that two dbextents are being defined.

MAXPOOLS

MAXPOOLS specifies the maximum number of storage pools that can ever be defined for the database. Its value can range from 1 to 999. The default is 32. In the example in Figure 94, MAXPOOLS is allowed to default to 32.

MAXEXTNT

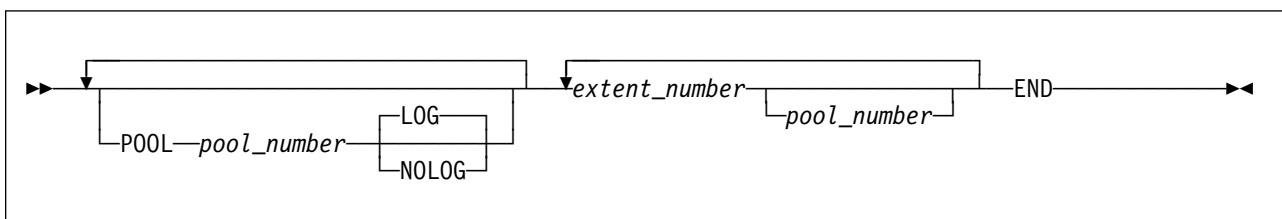
MAXEXTNT specifies the maximum number of dbextents that can ever be defined for the database. Its value can range between 1 and 999. The default is 64. In the example in Figure 94, MAXEXTNT is set to 200.

MAXDBSPC

MAXDBSPC specifies the maximum number of dbspaces that can ever be defined for the database. Its value can range from the number of dbspaces specified during database generation to 32 000. The default is 1 000. In the example in Figure 94, MAXDBSPC is explicitly set to 1 000.

Each keyword control statement can be specified on its own input record or multiple statements can be specified on one input record.

Specifying Initial Storage Pools and Dbextents: These keyword control statements identify the initial set of nonrecoverable storage pools and define the initial set of dbextents. The format for specifying these values is :



POOL

Include the POOL keyword control statement only for those storage pools you want to define as nonrecoverable: if you omit it, it will be defined as recoverable. You can specify as many nonrecoverable storage pools as you want up to the MAXPOOLS value. All POOL keyword control statements must precede the dbextent keyword control statements as shown. For more information, see “Nonrecoverable Storage Pools” on page 250.

pool_number

For *pool_number*, specify the number of the storage pool. You cannot specify 1: storage pool 1 is the default storage pool for dbspaces, so it cannot be defined as nonrecoverable.

LOG

The LOG option, which indicates that the storage pool is to be recoverable, is the default. Specify the NOLOG option if the storage pool is to be nonrecoverable.

extent_number/pool_number

The dbextent definition keyword control statements follow the POOL statements. They define an initial set of dbextents (by number), and the storage pool assignment for each. Any dbextent defined here must have a corresponding MDISK control statement for a VM minidisk. You must specify at least one dbextent for each storage pool that is referenced by the initial dspace definitions.

The first number in the pair is the extent number. You must define the dbextents in consecutive (numeric) order by extent number. Note that the extent numbers are decimal (unlike virtual device addresses, which are hexadecimal). The consecutive numbering is needed because the database manager requires the use of stylized ddnames in the CMS FILEDEF commands used to access the database. The SQLDBGEN EXEC creates these stylized ddnames in SQLFDEF for you. Both the SQLDBGEN EXEC and the database manager operate on the assumption that you are numbering the dbextents in consecutive (numeric) order. The stylized ddname format is DDSK n , where n is the extent number used in the dbextent keyword control statement.

The second number, which must be separated from the first by at least one blank, is the storage pool number. This is also a decimal value (as opposed to hexadecimal). If you do not specify the storage pool number, it defaults to 1.

Note: You cannot assign a dspace to a storage pool until a dbextent has been assigned there.

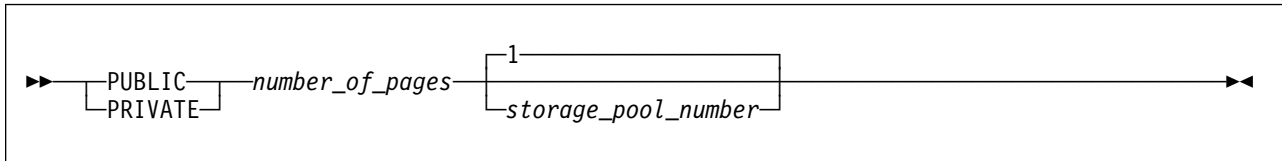
Each extent number/storage pool number pair must be entered on a separate input record. You can put comments on the dbextent keyword control statements by specifying the storage pool number and separating the comment from the number by at least one blank. A comment must be contained in the one input record for the dbextent: it cannot be continued on the next input record, which is interpreted as the next dbextent definition.

In the example in Figure 94, dbextent number 1 is assigned to storage pool number 1, and dbextent number 2 is assigned to storage pool number 2. The SQLDBGEN EXEC generates CMS FILEDEF commands in SQLFDEF of the following form:

```
FILEDEF DDSK1 DISK 32A...
FILEDEF DDSK2 DISK 32B...
```

Specifying Initial Dbspaces: These keyword control statements define the initial set of dbspaces, including public dbspaces that the database manager requires (system dbspaces), any user public and private dbspaces you need initially, and the internal dspace allocations for the database.

The format for specifying these values is :



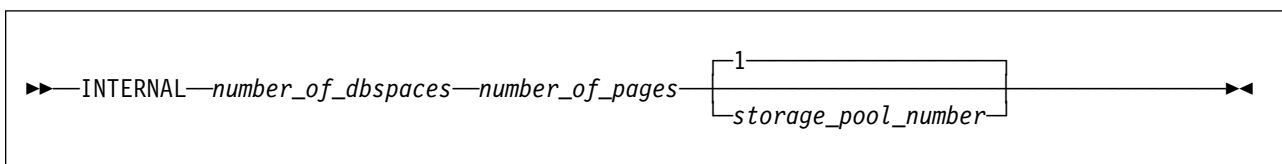
The *number of pages* value is the number of logical pages in the dbspace, rounded up to the next higher multiple of 128. The *storage_pool_number* must correspond to a pool that already has a dbextent, as defined by the dbextent keyword control statements.

You must define six public dbspaces for system use: the catalog, package, HELP text, ISQL tables, temporary install use, and the sample data tables dbspaces. In the example in Figure 94 on page 307 all are assigned to storage pool 1, but you can assign them elsewhere. The catalog and package dbspaces must always be assigned to a recoverable storage pool. In the example in Figure 94 on page 307, the first five dbspace keyword control statements specify:

- 12 800 pages for the catalog dbspace (SYS0001)
- 2 048 pages for the package dbspace (SYS0002)
- 8 192 pages for the HELPTTEXT dbspace
- 1 024 pages for the ISQL dbspace
- 512 pages for the sample dbspace.

The remainder of the public and private dbspaces shown in Figure 94 are user dbspaces of various sizes and storage pool assignments. After database generation finishes, the temporary installation dbspace is available as a user public dbspace. Thus, at the end of any database generation, there is at least one public dbspace of 1 024 pages available.

You must also define a number of internal dbspaces, for internal sorting and index creation. The general format for specifying the initial internal dbspace keyword control statement is:



This statement specifies the number (*number_of_dbspaces*) of equal size (*number_of_pages*) temporary dbspaces that you want. The *storage_pool_number* must correspond to a pool that already has a dbextent, as defined by the dbextent keyword control statements. The storage pool to which you assign the internal dbspaces can be either recoverable or nonrecoverable: if you do not specify the storage pool number, it defaults to 1. You must not delete all dbextents from this storage pool.

This internal dbspace keyword control statement must be the last dbspace definition input record before the END delimiter control statement. Separate the values in this statement by at least one blank.

In the example in Figure 94 on page 307, 80 internal dbspaces of 1 024 pages each are defined and assigned to storage pool 1.

Note: Because the catalog and package dbspaces are assigned to storage pool 1, performance is improved if internal dbspaces are assigned to some other recoverable storage pool. To keep the example simple, however, the internal dbspaces are assigned to storage pool 1.

You can change the specification of internal dbspaces on any ADD DBSPACE operation. For more information, see “Adding Dbspaces to the Database” on page 153.

Generally speaking, your input records for initial dspace definitions would follow the pattern shown in Figure 95.

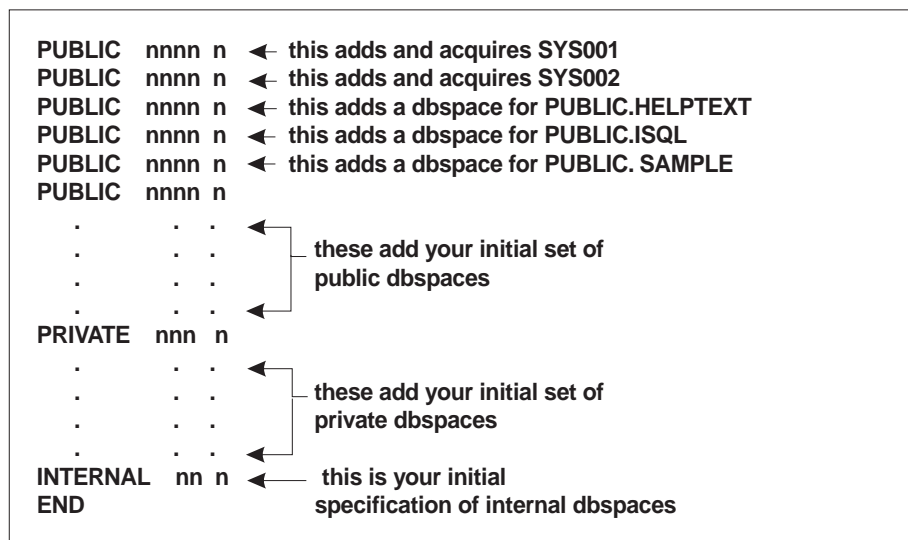


Figure 95. Input Records for Initial Dspace Definition

The first two dbspaces are public dbspaces (SYSTEM.SYS0001 and SYSTEM.SYS0002) that are both defined and acquired by the generation process for the catalog tables. You are advised to change either of these keyword control statements only if you want to define and allocate a larger dspace for the catalog tables or for packages, respectively. You do this by increasing the number of pages specified in the control statement.

There must also be keyword control statements for the three dbspaces that are acquired by the MACRO ARISDBU during SQLDBINS EXEC. These dbspaces are needed for the HELPTTEXT, ISQL, and sample dbspaces; the default sizes are 8192, 1024, and 512 pages respectively.

Note: The sample tables are loaded into the dspace PUBLIC.SAMPLE by the IBM-supplied DBS control file ARISAMDB MACRO V, which uses the user ID SQLDBA.

Other IBM-supplied EXECs call sample application programs, which manipulate the data in the sample tables. One sample program (and an EXEC to run it) is provided for each programming language that is supported by the database manager. Details of these programs are given in the *DB2 Server for VM Application Programming* manual.

If any of the above six database generation control statements are omitted, database generation may fail.

You can put comments on the dbspace keyword control statements if you specify the storage pool number and separate the comment from the storage pool number by at least one blank.

Loading the English version of the HELP text: The SQLDBINS EXEC issues the following prompt:

```
ARI6010D Do you want to install English DB2 Server for VM HELP text?  
Enter 0(No), 1(Yes), or 111(Quit)
```

1. If you respond 1 (Yes), the database manager does the following steps:
 - a. Creates HELP text tables
 - b. Creates indexes for all tables
 - c. Loads data into all HELP text tables
2. If you respond 0 (No), steps a and b are completed, but the tables are not loaded.
3. If you respond 111 (Quit), you immediately exit from the SQLDBINS EXEC, and the database is not generated. You can restart the SQLDBINS EXEC.

For information on installing HELP text in additional languages, see “National Language Support for Messages and HELP Text” on page 348.

Step 3: Optionally Changing the Application Server Default CHARNAME

The application server default CHARNAME value on a newly installed database manager is INTERNATIONAL (CCSID=500); on a migrated database manager, the default is ENGLISH (CCSID=37). To change the application server default CHARNAME (and with it the application server default CCSID, classification tables, and translation tables), specify the new CHARNAME as an SQLSTART EXEC parameter. For more information, see “Character Set Considerations at Startup” on page 62.

For information on creating a new CHARNAME, CCSID, and character set, see “National Language Support for Messages and HELP Text” on page 348.

Step 4: Optionally Changing the Application Server Default Character Subtype

If you use mixed data (data that contains both DBCS and SBCS characters), you may want to change the application server default character subtype to mixed. The application server default character subtype is the value used for new columns when the character subtype is not explicitly defined by the CREATE TABLE or ALTER TABLE statements, or supplied as a package option. The character subtype value is also used to determine whether the results of the CHAR, DIGITS, and HEX scalar functions and the character representation of date, time, or timestamp values, or special registers should be interpreted either as mixed data or as SBCS data.

The application server default character subtype is initially set to SBCS. To change it, subtype, see “Setting the Application Server Default Character Subtype” on page 342.

Step 5: Optionally Setting the DBCS Option to YES

If you are using a double-byte character set (DBCS), you should enable the DBCS option, which allows the database manager to correctly interpret SQL statements that contain DBCS strings. As a default, the DBCS option is not enabled. For information see “Using Double-Byte Character Set (DBCS)” on page 331.

Step 6: Changing the Password of User ID SQLDBA

One final task you should perform is to change the password of user ID SQLDBA in your new database. User ID SQLDBA is defined in all databases to have DBA authority. The password for this user ID is set to SQLDBAPW during database generation. Because this default password is common knowledge (it is in many product manuals), you should change it immediately after database generation. To do so, use ISQL, an application program, or the DBS utility to connect to the database as SQLDBA:

```
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW
```

Then change the password to one of your own choosing:

```
GRANT CONNECT TO SQLDBA IDENTIFIED BY newpw
```

Summary of Database Generation Process

Figure 96 summarizes the database generation process. It shows the major EXECs that are called, and key files that are created on the production minidisk.

SQLDBINS

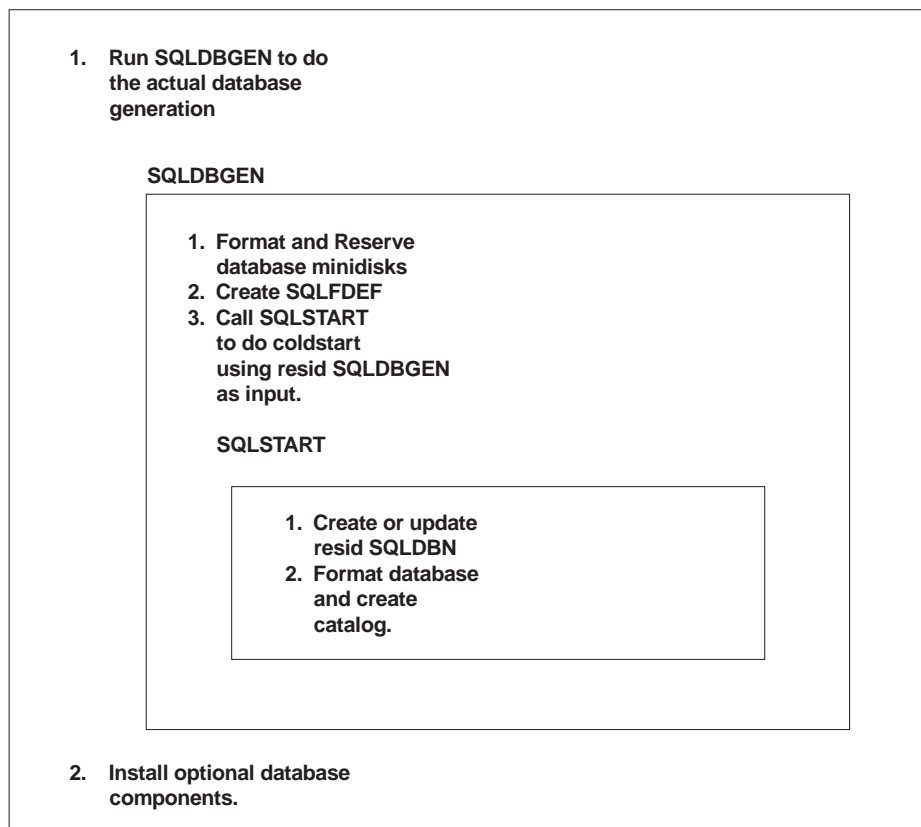


Figure 96. Summary of Database Generation Process

When you generate a database, be aware that:

- SQLFDEF files contain CP LINK and CMS FILEDEF commands, which are used when the database manager is started to access a database.
- SQLDBGEN files contain database generation control statements, which are used during database generation, and remain on the production minidisk to provide a record of the original database specifications.
- SQLDBN files identify the database, the database machine that last accessed the database, the addressing mode (AMODE) in which the database manager is running, and the DCSSID. They are used to aid in establishing communications between user machines and database machines.
- Optional activities are to change:
 - The application server default CHARNAME
 - The application server default character subtype
 - The DBCS option.
 - The password of SQLDBA (recommended).

VSE Guest Sharing Configuration

If you have VSE/ESA running as a guest under your VM operating system, VSE applications and users can access DB2 Server for VM databases through VSE guest sharing. The database accessed must be on the same VM system as the one that supports the VSE guest, or on another VM system in the same TSAF collection or SNA network. VM users and applications are not affected by VSE guest sharing.

The VSE guest sharing machine communicates with the database manager running under a VM/ESA operating system by using VSE APPCVM protocol, which is translated into the APPC/VM communication protocol. The following features are available to the VSE guest:

- Users and applications can access the database concurrently with VM users and applications.
- Multiple VSE guest machines can be supported by one database machine.
- If the VSE guest has its own database partition, it can access the database manager either on VSE or on VM. It cannot access an application server on VSE and VM in multiple user mode at the same time.

While VSE guest users are accessing an application server on VM, a batch application can also access an application server on VSE in single user mode. To do this, do **not** issue the SET APPCVM command for the batch application when you IPL VSE. Online users can access the database on VM by identifying it with the DBNAME parameter of the CIRB transaction. The batch application automatically accesses the application server on VSE.

- If the VM system is in a TSAF collection or an SNA network, VSE guest users and applications can access application servers on other processors in the collection or network.

The application server is identified to the VSE subsystem by an entry in the DBNAME directory and by the SET APPCVM command, which is issued when you IPL VSE. Batch/ICCF users access the application server identified by this command. Online users can access another application server by specifying it with the DBNAME parameter of the CIRB transaction. For more information on the SET APPCVM command and the CIRB transaction, see “Operating VSE Guest Sharing”

on page 103 and Chapter 5, “Operating the Online Support for VSE Guest Sharing” on page 103.

The user ID supplied by the CIRB transaction identifies CICS to the database machine. The user ID is the CICS APPLID, and defaults to DBDCCICS, unless you supply a different APPLID in the DFHSIT macro. For more information on CICS, see “Implicit CONNECT Support” on page 111.

Users who know the password for the CIRB transaction can have DBA authority in databases accessed by CIRB. The DB2 Server for VM user ID for CICS is set up by CIRB. Anyone accessing a database under that user ID has DBA authority.

The authority of users who know the CIRB transaction password can be limited using VM directory control statements. For more information on directory control statements, see “VM Directory Control Statements” on page 146.

VSE Guest Sharing Restrictions

VSE guests who access the database manager on the VM/ESA operating system are subject to the following restrictions:

- Guest users and applications cannot access the database manager on VM in single user mode.
- A record exists in the DBNAME directory, and the APPLID and DBNAME are identical.
- The APPLID (and DBNAME) does not begin with the characters “SYSARI.”

Examples of VSE Guest Sharing Configurations

The following figures show examples of VSE guest sharing configurations.

Figure 97 shows an example where both the VSE guests and the application server they access are on the same processor.

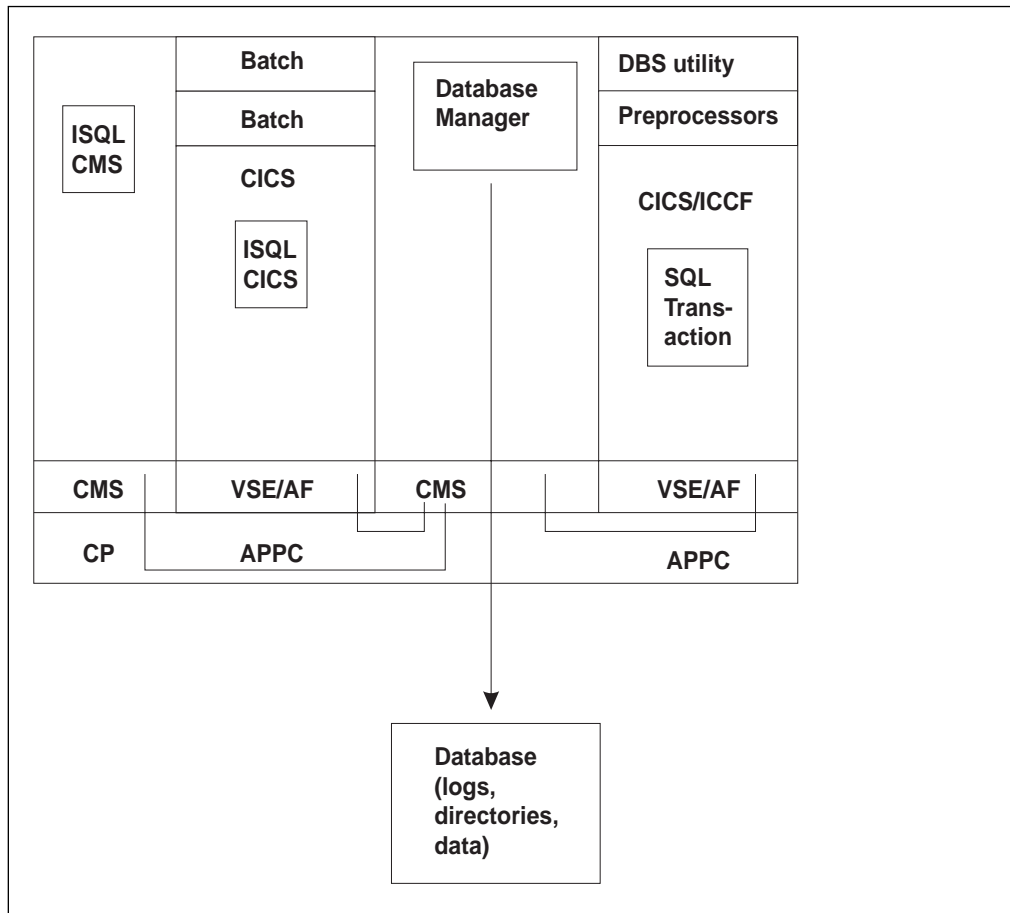


Figure 97. VSE Guest Sharing on One Processor

Figure 98 shows an example where the VSE guest and the application server it accesses are on different processors. Communication is handled by TSAF.

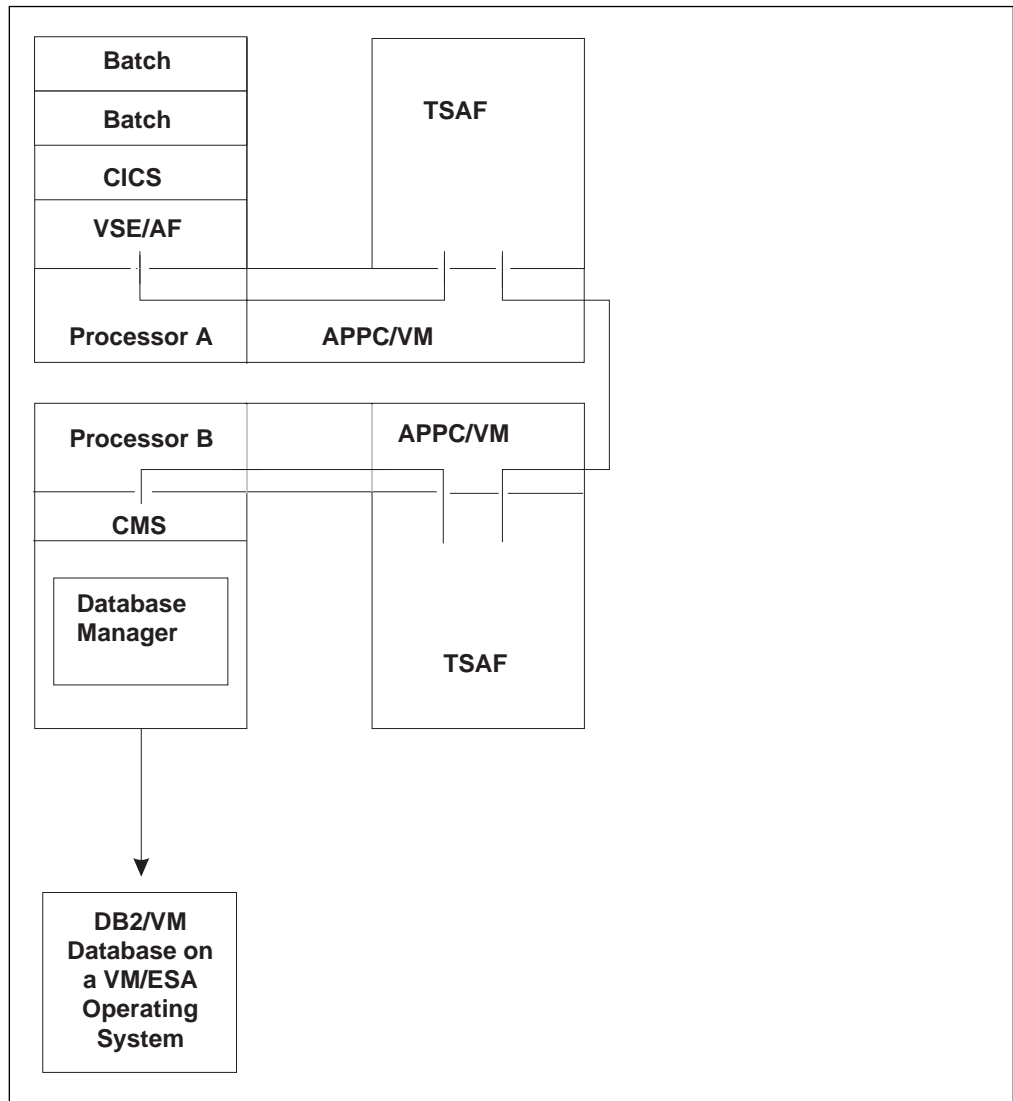


Figure 98. VSE Guest Sharing with TSAF

Figure 99 shows an example where the VSE guest and the database it accesses are on different processors. Communication is handled by the VTAM product and AVS over an SNA network.

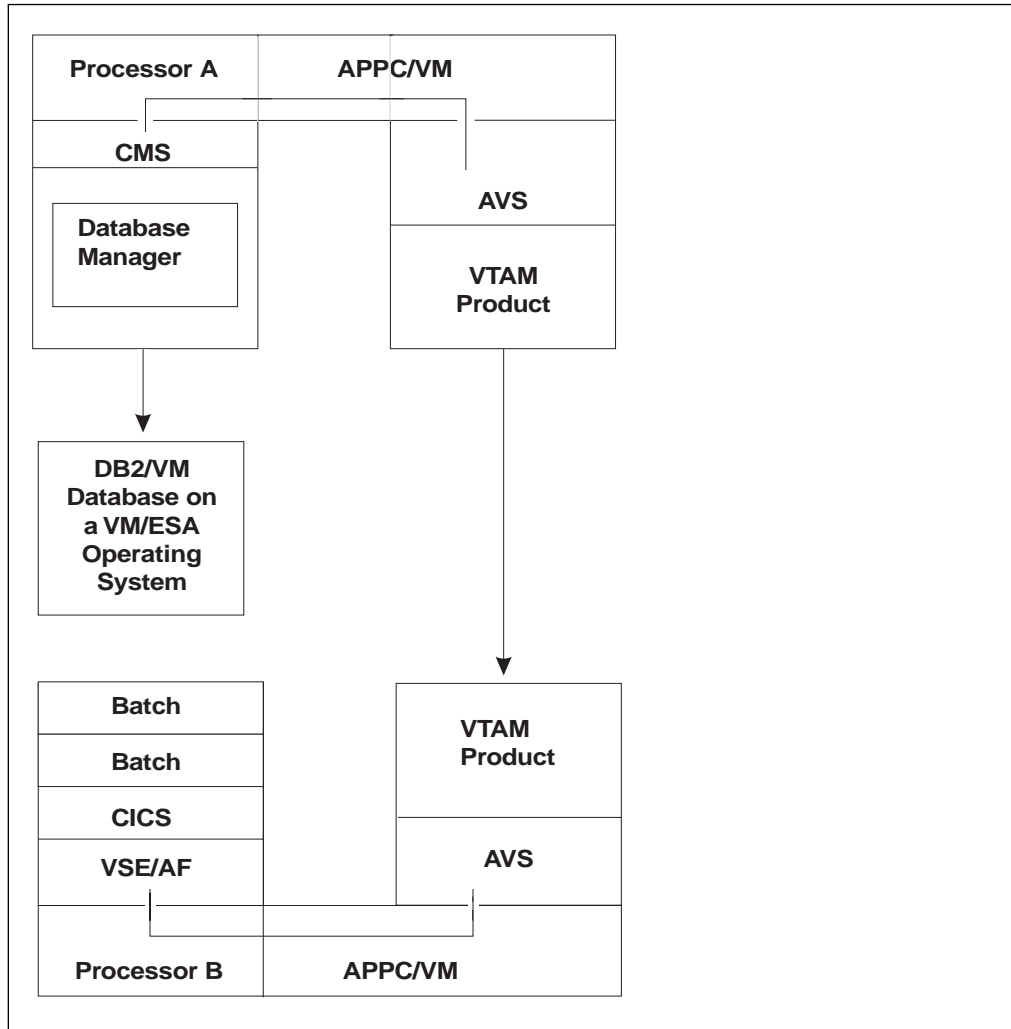


Figure 99. VSE Guest Sharing with VTAM Product and AVS

Chapter 13. Choosing a National Language and Defining Character Sets

A national language (as opposed to a programming language) is a language used in or by a nation. The database manager can work with data in national languages represented by a single-byte character set (SBCS) or by a double-byte character set (DBCS). The database manager will also support MBCS data from other platforms, which will be converted to SBCS and DBCS data. The following are some of the single-byte character sets that are shipped with the database manager:

- French
- English
- German
- Spanish
- Italian.

Examples of double-byte character sets that are shipped with the database manager include:

- Korean
- Japanese
- Chinese.

If you want a complete list of the character sets that are available, review the `SYSTEM.SYSCCHARSETS` catalog table.

This chapter describes the facilities that the database manager provides for national languages:

- `CHARNAME` specification

This facility allows you to specify character sets and CCSIDs other than the installation or migration default. The application server default `CHARNAME` for a new installation is `INTERNATIONAL` (CCSID=500). The application server default `CHARNAME` for a migrated system is `ENGLISH` (CCSID=37). The application requester default `CHARNAME` is always `INTERNATIONAL` (CCSID=500). The database manager can use alternative character sets for identifying character usage and for folding lowercase characters to uppercase.

This facility provides for the proper interpretation and use of national language characters not included in the default character set, for example, characters with umlauts, accents, and tildes.

This facility also provides for the proper interpretation of data from application requesters or application servers which use different character sets and code pages. Character conversion is performed on data when the CCSID of the application requester and the CCSID of the application server are different. The application server default CCSID is determined from the application server default `CHARNAME`.

It is very important that the application server and application requester have the same CCSID value unless there is a specific reason for them to be different. When the application server and application requester have different CCSID values, character conversion cannot be avoided. This conversion has an associated performance overhead, and causes performance degradation. CCSID conversion of data also affects the sargability of predicates. For more

information on performance, see the *DB2 Server for VSE & VM Performance Tuning Handbook* manual.

- DBCS option

This option, when it is set to YES, allows the database manager to correctly interpret the shift-out (X'0E') and shift-in (X'0F') characters that delimit EBCDIC DBCS strings. The DBCS option is set on both the application server and the application requester.

- Multiple language messages

The database manager provides multiple language message support to allow users to select the language in which error and informational messages appear (the language must already be installed). The operator can select the language for operator messages.

If an ISQL user selects a national language that is different from the national language set by the operator on the application server, when the ISQL user issues an operator command the output is in the language set by the operator.

For example, the operator sets the application server national language for operator messages to ENGLISH. The ISQL users set the application requester to KANJI. When the user issues an operator command, the result is in ENGLISH.

- Multiple language HELP text.

The database manager provides multiple language HELP text support. ISQL users can interactively retrieve help information on messages and codes and command reference information. The help facility allows ISQL users to retrieve help in the language of their choice (provided that the language version of the HELP text is installed). Information on DB2 Server for VM HELP text is contained in this chapter, and in the *DB2 Server for VM Database Administration* manual.

The database manager also provides graphic data types for use with strings of DBCS characters, as well as a mixed subtype for character data that contains both DBCS and SBCS characters. For more information about using graphic and mixed data, refer to the *DB2 Server for VSE & VM SQL Reference* manual.

Considerations when changing default CHARNAME and CCSID

If you are not using the default CHARNAME or CCSID during installation or migration, ensure you consider the following activities:

1. Choosing default CHARNAME and CCSID for the application server

- Installation - see "Choosing the Application Server Default CHARNAME and CCSID" on page 26.
- Migration - see "Choosing an Application Server Default CHARNAME" on page 35.

Note: Refer to "CCSID Conversion" on page 333 and "Determining CCSID Values" on page 337 for more information on CCSIDs.

2. Choosing default CHARNAME and CCSID for application requesters

- Installation - see "Choosing the Default CHARNAME and CCSID for Application Requesters" on page 29.

- Migration - see “Choosing the Default CHARNAME for All Application Requesters” on page 38.
3. Setting migration CCSID values
 - Installation - defaults are adequate.
 - Migration - see “Setting Migration CCSID Values” on page 36.
 4. Optionally, choosing application server default character subtype
 - Installation and migration - see “Choosing the Application Server Default Character Subtype” on page 28.
 5. Optionally, setting the DBCS option for the application server
 - Installation and migration - see “Setting the DBCS Option for the Application Server” on page 343.
 6. Optionally, setting the DBCS option for the application requester
 - Installation and migration - see “Setting DBCS Option for Application Requestors” on page 343.

Note: To understand the effect of DBCS options, refer to “Using Double-Byte Character Set (DBCS)” on page 331.

Changing from pre-Euro CHARNAME to Euro-compatible CHARNAME

DB2 Server for VSE & VM supports several code pages that are identical to existing code pages except that they include the Euro currency symbol rather than the International currency symbol (¤). If you choose to use a CHARNAME that corresponds to a CCSID for a code page that includes the Euro currency symbol, it is recommended that your existing character data that is currently tagged with a non-Euro CCSID be re-tagged with the corresponding Euro-compatible CCSID. These steps should only be used when changing your CCSID to the corresponding Euro-compatible CCSID as described in the following table:

Table 19. Non-Euro and Corresponding Euro-compatible CHARNAMEs and CCSIDs

From CCSID	From CHARNAME	To CCSID	To CHARNAME
37	English	1140	E-English
285	UK-English	1146	E-UK-English
297	French	1147	E-French
500	International	1148	E-International
273	German	1141	E-German
280	Italian	1144	E-Italian

Step 1: This step will locate character data that currently represents the International currency symbol (¤). The Euro-compatible code pages have replaced the International currency symbol with the Euro symbol. If your database does not contain character data that represents the International currency symbol, you can skip this step. If your data does contain character data that represents the International currency symbol, you must decide how to handle this. You can either skip this step, in which case character data that currently represents the International currency symbol will be interpreted as the Euro symbol once the CCSID is changed, or, you can change the character data that currently represents

the International currency symbol to some other value. The following SELECT statement will locate columns that are tagged with the non-Euro compatible CCSID.

```
SELECT CNAME,TNAME,CREATOR FROM SYSTEM.SYSCOLUMNS WHERE CCSID=current_ccsid
```

For each column found, run the following SELECT statement:

```
SELECT * FROM creator.tname WHERE cname LIKE'%cur_symbol%'
```

Note: If your keyboard cannot generate the International currency symbol, replace *cur_symbol* with the hex value X'9F'. In order to do this, you will have to use an editor that allows hexadecimal characters to be entered.

This command will show you all the rows for column *cname* that include the International currency symbol. You can now decide the appropriate value to change the International currency symbol to for each row.

Step 2: Re-tag existing character data with the new Euro-compatible CCSID.

This step makes use of three files (SQLCCSID SQLPARM, SQLCCSID EXEC, and SQLCCSID SYSIN) that are supplied by the DB2 Server for VSE & VM. Use these files as follows:

1. Shut down the database server.
2. Modify SQLCCSID SQLPARM if necessary. If you use any initialization parameters that must be specified when the database manager is started in single user mode, add them to the SQLCCSID SQLPARM file. Do **not** delete any lines currently in this file.
3. Use the instructions provided in the SQLCCSID MACRO file to make the necessary changes to that file.
4. Review the SQLCCSID EXEC and, if necessary, use the instructions provided in the EXEC to modify it.
5. Run the SQLCCSID EXEC. This EXEC starts the database manager in single user mode to change the CCSIDs.
6. Start the database server using your normal procedures.

Using Alternative Character Sets

When the database manager folds keywords and identifiers from lowercase to uppercase, or folds user-supplied data using the default TRANSLATE scalar function, it bases the folding on the default character set specified on the SQLSTART initialization parameter. For information on setting the default character set, refer to "Choosing an Application Server Default CHARNAME" on page 35.

Some characters in other national languages must be delimited by double quotation marks (") before they can be accepted in identifiers. The double quotation marks indicate that special characters are within the identifier. No characters within delimited identifiers are folded from lowercase to uppercase. To get proper folding of these characters and to allow them as part of an unquoted identifier, you can specify your own character set, which includes both classification and folding tables. Specify the CHARNAME parameter at startup to have the database manager use your character set as the default. You can then use characters such as o-umlaut or n-tilde in identifiers without the use of double quotation marks.

| For information on how to define your own character set, see Appendix E,
| “Defining Your Own Character Set” on page 475.

Hexadecimal Values of the Sample Character Sets

You will probably be able to use one of the IBM-supplied sample character sets without modification. This section shows the hexadecimal value that is used to represent each valid character. If your devices use those hexadecimal values for the indicated characters, you can use the IBM-supplied samples.

The ENGLISH character set is shown in Figure 100 on page 324. Only those characters that are identifiable by the database manager are shown. Any hexadecimal code that does not have a character assigned to it is unusable for DB2 Server for VM keywords or unquoted identifiers. Such characters are usable in quoted identifiers and in constants and, of course, can be stored in the database.

For example, many display devices using an English character set assign a cent sign (¢) to X'4A'. In Figure 100 on page 324, however, no character is shown for the value X'4A' meaning that X'4A' is unusable for DB2 Server for VM keywords or unquoted identifiers. If you want to put a cent sign in an identifier, you must use a delimited identifier.

Another example is the tilde (~). In most ENGLISH character sets, the tilde is represented by X'A1'. The matrix shows no entry for X'A1'. So, regardless of what X'A1' represents in your character set, you must use a delimited identifier.

These rules apply to the matrices for the other character sets as well. An important point to remember is that the absence of a character in one of the matrices does not prevent you from using that character set. The characters are not undefined to the database manager; they merely have limited use (as described above). Often, this limited use is exactly how you want the hexadecimal code to be handled. Independent of this qualification, you should almost always be able to find a CCSID that you can use at your installation. When you decide on a CCSID, try to avoid using a non-standard CCSID to prevent possible problems in the future (such as the inability to connect to other application servers because they do not support your CCSID). If you require a CCSID that is not supplied in the catalog tables, check the *Character Data Representation Architecture Level 1, Registry* manual for other predefined registered CCSIDs.

Hex 1 Bits 4567		00				01				10				11				Hex 0
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0					SP	&	-						^				0
0001	1							/		a	j			A	J			1
0010	2									b	k	s		B	K	S		2
0011	3									c	l	t		C	L	T		3
0100	4									d	m	u		D	M	U		4
0101	5									e	n	v		E	N	V		5
0110	6									f	o	w		F	O	W		6
0111	7									g	p	x		G	P	X		7
1000	8									h	q	y		H	Q	Y		8
1001	9									i	r	z		I	R	Z		9
1010	A						!		:									
1011	B					.	s	.	#									
1100	C					<	*	%	@									
1101	D					()	-	'									
1110	E					+	;	>	=									
1111	F							?	"									

Figure 100. ENGLISH Character Set (CCSID=37)

The sample FRENCH character set is shown in Figure 101 on page 325. Translation from lowercase to uppercase is done as follows:

X'6A' is translated to X'E4'
 X'7C' X'C1'
 X'C0' X'C5'
 X'D0' X'C5'
 X'E0' X'C3'

These characters can be used in unquoted identifiers.

When evaluating the character set for use in your installation, remember that hexadecimal values that do not have characters assigned to them in Figure 101 on page 325 can be used in quoted identifiers.

Hex 1 Bits 4567		00				01				10				11				Bits 0,1 2,3 Hex 0			
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11				
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F				
0000	0					SP	&	-										é	è	ç	0
0001	1							/		a	j			A	J						1
0010	2									b	k	s		B	K	S					2
0011	3									c	l	t		C	L	T					3
0100	4									d	m	u		D	M	U					4
0101	5									e	n	v		E	N	V					5
0110	6									f	o	w		F	O	W					6
0111	7									g	p	x		G	P	X					7
1000	8									h	q	y		H	Q	Y					8
1001	9									i	r	z		I	R	Z					9
1010	A							û	:									-			
1011	B					.	s	,	£												
1100	C					<	*	%	à												
1101	D					()	_	·												
1110	E					+	:	>	=												
1111	F					!	^	?	"												

Figure 101. FRENCH Character Set (CCSID=297)

The sample GERMAN character set is shown in Figure 102 on page 326. Translation from lowercase to uppercase is done as follows:

X'4A' is translated to X'4A'
 X'5A' X'5A'
 X'6A' X'E0'
 X'A1' X'A1'
 X'C0' X'4A'
 X'D0' X'5A'
 X'E0' X'E0'

These characters can be used in unquoted identifiers.

When evaluating the sample character set for use in your installation, remember that hexadecimal values that do not have characters assigned to them in Figure 102 on page 326 can be used in quoted identifiers.

Some translation from lowercase to uppercase does not cause a change in the hexadecimal value. For more information, see "Step 3: Determine Translation Characters" on page 486.

Hex 1 Bits 4567		00				01				10				11				Bits 0,1 2,3 Hex 0
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0					SP	&	-						ä	ü	Ö	0	
0001	1							/		a	j	ß		A	J		1	
0010	2									b	k	s		B	K	S	2	
0011	3									c	l	t		C	L	T	3	
0100	4									d	m	u		D	M	U	4	
0101	5									e	n	v		E	N	V	5	
0110	6									f	o	w		F	O	W	6	
0111	7									g	p	x		G	P	X	7	
1000	8									h	q	y		H	Q	Y	8	
1001	9									i	r	z		I	R	Z	9	
1010	A					Ä	Ü	ö	:					-				
1011	B					.	s	,	#									
1100	C					<	*	%	\$									
1101	D					()	_	'									
1110	E					+	;	>	=									
1111	F					!	^	?	"									

Figure 102. GERMAN Character Set (CCSID=273)

The sample ITALIAN character set is shown in Figure 103 on page 327. Translation from lowercase to uppercase is done as follows:

X'5A' is translated to X'C5'
 X'6A' X'D6'
 X'79' X'E4'
 X'A1' X'C9'
 X'C0' X'C1'
 X'D0' X'C5'
 X'E0' X'C3'

These characters can be used in unquoted identifiers.

When evaluating the sample character set for use in your installation, remember that hexadecimal values that do not have characters assigned to them in Figure 103 on page 327 can be used in quoted identifiers.

Hex 1 Bits 4567		00				01				10				11				Hex 0
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0					SP	&	-						à	è	ç	0	
0001	1							/		a	j	ì		A	J		1	
0010	2									b	k	s		B	K	S	2	
0011	3									c	l	t		C	L	T	3	
0100	4									d	m	u		D	M	U	4	
0101	5									e	n	v		E	N	V	5	
0110	6									f	o	w		F	O	W	6	
0111	7									g	p	x		G	P	X	7	
1000	8									h	q	y		H	Q	Y	8	
1001	9							ù		i	r	z		I	R	Z	9	
1010	A						é	ò	:					-				
1011	B					.	s	,	£									
1100	C					<	*	%	à									
1101	D					()	-	·									
1110	E					+	;	>	=									
1111	F					!	^	?	"									

Figure 103. ITALIAN Character Set (CCSID=280)

The sample KATAKANA character set is shown in Figure 104 on page 328.

When evaluating the sample character set for use in your installation, remember that hexadecimal values that do not have characters assigned to them in Figure 104 on page 328 can be used in quoted identifiers.

Hex 1	Hex 0	00				01				10				11				First Hex Char. Bits 0,1	
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11		2,3
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0000	0					SP	&	-			リ		^					0	
0001	1						エ	/	i	ア	タ			A	J			1	
0010	2						オ	a	j	イ	チ	ハ		B	K	S		2	
0011	3						パ	b	k	ウ	ツ	ホ	t	C	L	T		3	
0100	4						ユ	c	l	イ	テ	マ	u	D	M	U		4	
0101	5						ヨ	d	m	エ	ト	ミ	v	E	N	V		5	
0110	6					ヲ	ツ	e	n	カ	ナ	ル	w	F	O	W		6	
0111	7					ア		f	o	キ	ニ	メ	x	G	P	X		7	
1000	8					イ	ー	g	p	ク	ヌ	モ	y	H	Q	Y		8	
1001	9					ウ		h		ケ	ネ	ブ	z	I	R	Z		9	
1010	A						!		:	コ	ノ	ゾ	レ						
1011	B					.	¥	,	#	q	r	s	□						
1100	C					<	*	%	@	サ		ヨ	フ						
1101	D					()	_	.	シ	ハ	ラ	ン						
1110	E					+	:	>	=	ス	ヒ	リ	ハ						
1111	F						.	?	"	ビ	フ	ル	.						

Figure 104. JAPANESE (Katakana) Character Set (CCSID=290, the SBCS Component of CCSID 5026)

The sample SPANISH character set is shown in Figure 105 on page 329. Translation from lowercase to uppercase is done as follows:

X'6A' is translated to X'7B'

These characters can be used in unquoted identifiers.

When evaluating the sample character set for use in your installation, remember that hexadecimal values that do not have characters assigned to them in Figure 105 on page 329 can be used in quoted identifiers.

Hex 1 Bits 4567	Hex 0	00				01				10				11				Bits 0,1 2,3
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0					SP	&	-									0	
0001	1							/		a	j			A	J		1	
0010	2									b	k	s		B	K	S	2	
0011	3									c	l	t		C	L	T	3	
0100	4									d	m	u		D	M	U	4	
0101	5									e	n	v		E	N	V	5	
0110	6									f	o	w		F	O	W	6	
0111	7									g	p	x		G	P	X	7	
1000	8									h	q	y		H	Q	Y	8	
1001	9									i	r	z		I	R	Z	9	
1010	A							ñ	:					^				
1011	B					.	Pts	,	Ñ					!				
1100	C					<	*	%	@									
1101	D					()	-	'									
1110	E					+	:	>	=									
1111	F							?	"									

Figure 105. SPANISH Character Set (CCSID=284)

Specifying an IBM-Supplied Character Set at Run Time

If the hexadecimal codes in one of the sample character sets matched those used by your devices, you can specify the character set at run time. To use a character set, specify the CHARNAME parameter when starting the application server. The CHARNAME parameter is valid in both single and multiple user mode. For information on how to specify the CHARNAME parameter, see “Setting the Application Server Default CHARNAME and CCSIDs” on page 338. Examples of IBM-supplied sample character sets are:

- ARABIC
- CYRILLIC
- DANISH-NORWEGIAN
- E-INTERNATIONAL
- ENGLISH
- ESTONIAN
- FINNISH-SWEDISH
- FRENCH
- GERMAN
- GREEK
- GREEK-423

- HEBREW
- ICELANDIC
- INTERNATIONAL
- ITALIAN
- JAPANESE-ENGLISH
- KATAKANA
- KOREAN
- LAO
- S-CHINESE
- SPANISH
- T-CHINESE
- THAI
- UK-ENGLISH
- UKRAINIAN
- VIETNAMESE
- 290
- 833
- 836
- 870
- 930
- 939
- 1027
- 1112
- 28709.

Figure 106 shows an example of starting the application server. The CHARNAME parameter indicates that the database manager is to use the FRENCH sample character set, and the default CCSID of 297.

```
SQLSTART DB(SQLDBA) ID(MYBOOT) PARM(PARMID=WARM1,CHARNAME=FRENCH)
```

Figure 106. Starting the Application Server to Use the FRENCH Character Set

The default character sets ENGLISH (CCSID=37) and INTERNATIONAL (CCSID=500) are hardcoded into this product. For example, if you specify ENGLISH for the CHARNAME parameter, the database manager uses the ENGLISH character set that is coded internally. The internally coded character set is used even if a row exists in SYSTEM.SYSCHARSETS that has ENGLISH or INTERNATIONAL in its NAME column. (Neither the sample ENGLISH character set nor the sample INTERNATIONAL character set is used, although you can load either into SYSTEM.SYSCHARSETS. They are provided to make the definition of your own character sets easier.)

If you specify the name of a character set that is not defined in SYSTEM.SYSCHARSETS, the database manager displays an error message and uses the character set that was specified previously. If the character set is defined incorrectly in SYSTEM.SYSCHARSETS, an error message is displayed, and the database manager uses the character set that was previously specified.

Using Double-Byte Character Set (DBCS)

The double-byte character set (DBCS) option supports the use of DBCS characters in identifiers, constants and data. Identifiers can be either:

- Host identifiers (such as host variables), or
- SQL identifiers (such as dbspaces, tables or columns).

Constants and data containing DBCS characters can be either:

- Graphic data, or
- Character data with a mixed subtype (that is, character data containing DBCS characters).

Setting the DBCS option also ensures that:

- A shift-out character is paired with a shift-in character on output,
- When mixed character data is truncated, truncation does not occur between the two bytes of a DBCS character.

If your installation uses a double-byte character set, you should consider setting the DBCS option to YES. The DBCS option can be set for the application server, the application requester, or both. For the application server, the DBCS option is set in the SYSTEM.SYSOPTIONS catalog table. For the application requester, the DBCS option is set by the DBCS parameter of the SQLINIT EXEC.

For information on enabling the DBCS option, see “Setting the DBCS Option for the Application Server” on page 343, and “Setting DBCS Option for Application Requestors” on page 343.

When the DBCS option is set to YES, the shift-out (X'0E') and shift-in (X'0F') delimiters are recognized in both identifiers in SQL statements and mixed data character string constants. The recognition of the delimiters provides the following benefits:

- On the application server
 - SQL identifiers can contain DBCS characters.
- On the application requester
 - Host identifiers can contain DBCS characters.
 - The DBS utility processing ensures the pairings of shift-out and shift-in characters.
 - ISQL allows the input, print, and display of DBCS characters and mixed data.

However, setting the DBCS option induces overhead for checking the proper pairing of shift-out and shift-in characters.

Identifiers Containing DBCS Characters

Identifiers can be either host identifiers or SQL identifiers.

To use host identifiers that contain DBCS characters, the VSE application requester must have the DBCS option set to YES. (For more information on the SQLINIT EXEC, see the *DB2 Server for VM Database Administration* manual.)

To use ordinary SQL identifiers that contain DBCS characters, the application server must have the DBCS option in the SYSTEM.SYSOPTIONS catalog table set to YES, and must also support DBCS characters and mixed data. The application server supports DBCS characters and mixed data when a mixed CHARNAME is specified as an initialization parameter. A mixed CHARNAME has a non-zero value for the CCSIDMIXED row in the SYSTEM.SYSOPTIONS catalog table. For more information, see “Choosing the Application Server Default CHARNAME and CCSID” on page 26.

If the DBCS option is set to YES for the application server, you can use DBCS characters in ordinary SQL identifiers. The identifier can be DBCS characters, or can contain a DBCS substring.

Identifiers are recorded in the catalog tables. When the database manager stores identifiers that contain DBCS characters, it also stores the shift-out and shift-in delimiters. The delimiters are stored because all columns of the catalog tables that contain identifiers have a data type of either CHAR or VARCHAR.

The number of bytes required to represent a string of DBCS characters is equal to:

$$2 \times \text{the number of DBCS characters} + 2$$

For more information on how identifiers are used in application programs, see the *DB2 Server for VM Application Programming* manual.

Constants and Data Containing DBCS Characters

Constants and data containing DBCS characters can be either graphic data or character data with a mixed subtype.

To use graphic and mixed constants or data, the application server and the application requester must support mixed data. This support is provided by specifying a mixed CHARNAME on the SQLSTART EXEC for the application server, and on the SQLINIT EXEC for the application requester. The application server supports graphic and mixed data when the default CHARNAME is a mixed CHARNAME. A mixed CHARNAME has a non-zero value for the CCSIDMIXED row in the SYSTEM.SYSOPTIONS catalog table. For more information, see “Choosing the Application Server Default CHARNAME and CCSID” on page 26. The application requester supports graphic and mixed data when a mixed CHARNAME is specified for the SQLINIT EXEC. (For more information, see the *DB2 Server for VM Database Administration* manual.) Using a mixed CHARNAME provides the following benefits:

- On the application server
 - Character string constants can contain mixed data consisting of DBCS and SBCS characters.
 - When character string constants containing DBCS characters are used in SQL statements, the data is correctly interpreted as having a mixed subtype and a mixed CCSID.
- On the application requester
 - Character string constants can contain mixed data consisting of DBCS, SBCS, and on some platforms, MBCS characters. (The MBCS characters will be stored as SBCS or DBCS characters by the database manager.)

If the DBCS value for the application server is set to YES in the SYSTEM.SYSOPTIONS catalog table, the SQLINIT DBCS option for the application requester does not have to be YES for mixed data to be recognized. Mixed data is still recognized and tagged with the appropriate mixed CCSID if the SQLINIT CHARNAME is a mixed CHARNAME, and the character string constant is a valid mixed string.

The implied data type of all string character constants is VARCHAR. When the DBCS option is set to YES:

- Constants with only SBCS characters have a subtype of SBCS.
- Constants with a combination of SBCS and DBCS characters have a subtype of mixed. For example:

```
'abc<XXYYZZ>'
```

- Constants containing only DBCS are also of the character data type with a subtype of mixed. They are *not* considered to be graphic constants. For example:

```
'<XXYYZZ>'
```

When the DBCS option for the application server is set to NO, all character constants have a subtype of SBCS.

CCSID Conversion

Internally, character data is stored as hexadecimal values called code points. When a device interprets or displays a code point as a character, it uses a code page, which is a set of assignments of characters to code points. If two terminals use different code pages, they can display the same code point as a different character. For example, in code page 37, the code point X'4F' represents a vertical bar (|), but in code page 500, the code point X'4F' represents an exclamation mark (!).

As of Version 3 Release 3, the database manager supports Coded Character Set Identifiers (CCSIDs). The CCSID attribute specifies which code page to use both to map code points to characters, and to map characters to code points. The CCSID and the code points are used together to determine the character that the code point represents.

Note: The default CCSID is implicitly set by the default CHARNAME.

For example, suppose a DB2 Server for VM application requester has the default CHARNAME set to ENGLISH (CCSID=37), the DB2 Server for VM application server has the default CHARNAME set to INTERNATIONAL (CCSID=500) and the PROTOCOL parameter on both the application requester and the application server is *not* SQLDS. In this case, if the user inserts an exclamation mark into a character column, the application requester sends X'5A' (the code point that represents an exclamation mark in the code page used with CCSID 37). The application server converts X'5A' to X'4F' (because X'4F' represents an exclamation mark in the code page used with CCSID 500), then stores X'4F' in the column.

If another application requester using a different CCSID retrieves the character, X'4F' is converted to the code point that represents an exclamation mark in the code page specified by the application requester CCSID. The character is

interpreted and displayed correctly, and the hexadecimal value that is stored in the database is not changed.

For more information on how to decide the default CCSID values you should use, see “Determining CCSID Values” on page 337.

The sections that follow discuss the following topics:

- How to determine the CCSID values
- How to set the application server default CHARNAME and CCSID values
- How to set the application requester default CHARNAME and CCSID values
- How to set the application server default character subtype
- How to set the DBCS option for application requesters and application server.
- EUC Conversion.

For examples that show the interactions among the different values, see “Examples of Setting Values for an Installation” on page 345.

If an application requester and an application server do not use the same default CCSID, CCSID conversion is done during communications between the two.

Note: For an application requester using an ASCII representation of the data, CCSID conversion **always** occurs.

Table 20. CCSID Conversion between the Application Server and Application Requester

Application Requester		Application Server		
		DB2/VM		Non-DB2/VM (incl. DB2/VSE)
		SQLDS	AUTO	
DB2/VM	SQLDS	NO	NO	Not Allowed
	AUTO	NO	YES	YES
	DRDA	Not Allowed	YES	YES
Non-DB2/VM		Not Allowed	YES	Not Applicable

Note:

In the figure:

- “NO” indicates that the application requester CCSIDs are not recognized and CCSID conversion is not done between the application server CCSIDs and the application requester CCSIDs.
- “YES” indicates that the application requester CCSIDs are recognized by the application server, and CCSID conversion is active between the application server CCSIDs and the application requester CCSIDs.
- “Not Allowed” indicates that this combination of PROTOCOL parameters is not supported.

- “non-DB2/VM” is a non-DB2/VM application server or application requester that supports the DRDA protocol. In the case of application servers, it also includes DB2/VSE. (DB2/VSE cannot function as an application requester.)

Table 21 and Table 22 on page 337 show CHARNAMEs and the corresponding CCSIDs that can be used as system defaults. Table 21 shows the SBCS CHARNAME CCSIDs, and Table 22 on page 337 shows the mixed CHARNAME CCSIDs, with the component SBCS and DBCS CCSIDs for each mixed CCSID.

Table 21 (Page 1 of 2). SBCS CCSIDs

CCSID	Character Set	Code Page	CHARNAME	Description
37	697	37	ENGLISH	Country extended code pages (CECP): USA, Canada (S/370* system), Netherlands, Portugal, Brazil, Australia, New Zealand
273	697	273	GERMAN	CECP: Austria, Germany
277	697	277	DANISH-NORWEGIAN	CECP: Denmark, Norway
278	697	278	FINNISH-SWEDISH	CECP: Finland, Sweden
280	697	280	ITALIAN	CECP: Italy
284	697	284	SPANISH	CECP: Spain, Latin America (Spanish)
285	697	285	UK-ENGLISH	CECP: United Kingdom
290	1172	290	290	Japanese Katakana, extended host single byte
297	697	297	FRENCH	CECP: France
420	235	420	ARABIC	Arabic (all presentation shapes)
423	218	423	GREEK-423	Greek (Coexistence)
424	941	424	HEBREW	Hebrew
500	697	500	INTERNATIONAL	CECP: Belgium, Canada (AS/400* system), Switzerland, International Latin-1
833	1173	833	833	Korean, extended host single byte
836	1174	836	836	Simplified Chinese, extended host single byte
838	1176	838	THAI	Thai, extended host single byte
870	959	870	870	ROECE (Regional Office for East & Central Europe) Latin-2 Multilingual
871	697	871	ICELANDIC	CECP: Iceland
875	925	875	GREEK	Greek
1025	1150	1025	CYRILLIC	Cyrillic Multilingual Turkish Latin 5
1027	1172	1027	1027	Japanese Latin, extended host single byte
1112	1305	1112	1112	Latvian/Lithuanian
1122	1307	1122	ESTONIAN	Estonian
1123	1326	1123	UKRAINIAN	Cyrillic Ukrainian EBCDIC

Table 21 (Page 2 of 2). SBCS CCSIDs

CCSID	Character Set	Code Page	CHARNAME	Description
1130	1336	1130	VIETNAMESE	EBCDIC Vietnamese
1132	1341	1133	LAO	EBCDIC Lao
1148	697	500	E-INTERNATIONAL	International Euro CECP
1140	697	37	E-ENGLISH	English Euro CECP
1141	697	273	E-GERMAN	German Euro CECP
1144	697	280	E-ITALIAN	Italian Euro CECP
1146	697	285	E-UK-ENGLISH	UK English Euro CECP
1147	697	297	E-FRENCH	French Euro CECP
28709	1175	37	28709	Traditional Chinese, extended host single byte

Mixed	Component CCSIDs	Character Set	Code Page	CHARNAME	Description
930	290 (SBCS) 300 (DBCS)	1172 1001	290 300	930	Japanese (Katakana)-Kanji mixed host (including 4370 user-defined characters) extended single byte
933	833 (SBCS) 834 (DBCS)	1173 934	833 834	KOREAN	Korean host mixed (including 1880 user-defined characters) extended single byte
935	836 (SBCS) 837(DBCS)	1174 937	836 837	S-CHINESE	Simplified Chinese host mixed (1880 user-defined characters) extended single byte
937	28709 (SBCS) 835 (DBCS)	1175 935	37 835	T-CHINESE	Traditional Chinese host mixed (6204 user-defined characters) extended single byte
939	1027 (SBCS) 300 (DBCS)	1172 1001	1027 300	939	Japanese (Latin)-Kanji mixed host (including 4370 user-defined-characters) extended single byte
1364	833 (SBCS) 834 (DBCS)	65535 65535	833 834	KOREAN-1364	Korean host mixed extended including 11,172 full hangul
1388	836 (SBCS) 837 (DBCS)	65535 65535	846 837	S-CHINESE-GBK	S-Ch DBCS-Host Data GBK mixed, all GBK character set and other growing chars
5026	290 (SBCS) 4396 (DBCS)	1172 370	290 300	KATAKANA	Japanese (Katakana)-Kanji mixed host (including 1880 user-defined characters) extended single byte
5035	1027 (SBCS) 4396 (DBCS)	1172 370	1027 300	JAPANESE-ENGLISH	Japanese (Latin)-Kanji mixed host , (including 1880 user-defined characters) extended single byte

For more information about CCSIDs, see the *Character Data Representation Architecture Level 1, Registry*, and the *Character Data Representation Architecture Reference and Registry* manuals.

For information on the types of DBCS conversion that can be done between CCSIDs, see “Coding Your Own TRANSPROC Exit” on page 380.

Determining CCSID Values

When displaying characters on your terminal display, the default CCSID values for an application requester must be compatible with the code page that was used to generate its terminal controller. If the defaults are incompatible, characters will not be displayed or interpreted as expected, and the results of queries or inserts of either character or graphic data will be unreliable. Table 21 on page 335 and Table 22 show the code pages that are compatible with each CCSID, and the CHARNAME value that you should specify. For example, if the controller was generated with code page 37, you should specify ENGLISH for the CHARNAME

parameter. This value sets the value of CCSIDSBBCS to 37, and the values of CCSIDMIXED and CCSIDGRAPHIC to 0. You can use the SQLGLOB EXEC to set default values for all application requesters. For more information, see “Setting the Default CHARNAME and CCSIDs for All Application Requesters” on page 341 . The system-wide defaults may not be suitable for all application requesters: some may have a controller generated to use a code page incompatible with the system-wide default CHARNAME. In this situation, you should set the CHARNAME parameter for individual application requesters. For more information on this topic, see “Setting the Application Requester Default CHARNAME and CCSIDs” on page 341.

The default CCSID values for the application server can be set to any value you want, but keep in mind that you may want to set the default CCSIDs for the application server to values that can be used as defaults by the majority of the application requesters. This can reduce the amount of CCSID conversion that will be necessary.

In particular, the CHARNAME INTERNATIONAL (CCSID=500) warrants special attention. This CHARNAME is composed of a code page that supports all the characters that are supported by the Latin-1 countries, including Australia, Austria, Belgium, Brazil, Denmark, Canada, the Faroe Islands, Finland, France, Germany, Hong Kong, Iceland, Italy, Japan, the Netherlands, New Zealand, Norway, Portugal, Spain, Latin America, Sweden, Switzerland, the United Kingdom, and the United States. If all application requesters and application servers in these countries use this CCSID, then single-byte CCSID conversion will not be necessary for accessing data from different sites. This may provide savings because of lower CPU usage.

Often, it may not be appropriate to use CCSID 500 at every site. For example, you may have to use existing equipment (such as terminal controllers that use other character sets and code pages), or you may have a large quantity of data that is stored using a CCSID other than 500. However, if you plan to frequently access data from other countries, you should consider migrating your data and hardware to CCSID 500, both for performance reasons, and for the ease of data management.

Setting the Application Server Default CHARNAME and CCSIDs

The different uses of the default system CCSIDs are shown in “Choosing the Application Server Default CHARNAME and CCSID” on page 26. Data in columns which were migrated from a release earlier than Version 3 Release 3 have a CCSID which is obtained from rows in the SYSTEM.SYSOPTIONS catalog table. These rows are: MCCSIDSBBCS, MCCSIDMIXED and MCCSIDGRAPHIC. For more information on the SYSTEM.SYSOPTIONS catalog table, see the *DB2 Server for VSE & VM SQL Reference* manual. To change either the application server default CCSIDs or the CCSIDs that are used for data in migrated columns, you must have DBA authority.

The only way to change the application server default CCSID is to change the application server default CHARNAME. This is done by specifying the CHARNAME parameter of the SQLSTART EXEC the next time the application server is started. This also updates the following columns of the SYSTEM.SYSOPTIONS catalog table: CCSIDSBBCS, CCSIDMIXED and CCSIDGRAPHIC. For more information, see “Character Set Considerations at Startup” on page 62.

You may have to use different default CCSIDs for columns that were created before the migration than for columns created after the migration. For example, suppose that you are migrating your database and want to use the INTERNATIONAL character set (CCSID=500) for character columns that were created after the migration. Character columns that existed before migration were created with the ENGLISH character set (CCSID=37). To ensure that the data in existing character columns is displayed and interpreted correctly, (that is, as was done before the migration), you require an MCCSIDSBACS value of 37, and a CCSIDSBACS value of 500.

Be very careful when using different default CCSIDs. This should only be done when there is a specific reason for them to be different. When the application server and application requester have different CCSID values, character conversion cannot be avoided. This conversion has an associated performance overhead, and causes performance degradation. CCSID conversion of data also affects the sargability of predicates. For more information on performance, see the *DB2 Server for VSE & VM Performance Tuning Handbook*.

Note: Use caution when you change the application server default CCSIDs. For more information, see “Determining CCSID Values” on page 337.

For many characters, the corresponding hexadecimal value in the International code page is the same as in the English code page. However, this is not true of all characters. For example, in the English code page the hexadecimal value corresponding to the exclamation mark (!) is '5A', but in the International code page the value is '4F'. Table 23 lists the differences between the International code page and the English code page.

Character	CCSID=37	CCSID=500
^	X'B0'	X'5F'
¢	X'4A'	X'B0'
!	X'5A'	X'4F'
[X'BA'	X'4A'
]	X'BB'	X'5A'
	X'4F'	X'BB'
¬	X'5F'	X'BA'

For more information on code page details, see the *Character Data Representation Architecture Level 1, Registry* manual.

Columns must be tagged with the CCSID that corresponds to the code page with which they were created or the results of queries on these columns will be unreliable. For example, suppose that the following column was created with the English character set before migration:

```
CHARDATA
-----
ABCDEFGH
  kjp!
  ¬ds!
```

If MCCSIDSBBCS is 37 (corresponding to English), and CCSIDSBBCS is 500 (corresponding to International), performing a SELECT operation on this column after the migration gives the results shown above. However, if MCCSIDSBBCS is incorrectly set to 500 (corresponding to the International character set), performing a SELECT operation on the column produces the following result:

```
CHARDATA
-----
ABCDEFGH
  kjp]
  ^ds]
```

In this example, to ensure reliable results, MCCSIDSBBCS must be 37, regardless of the value of CCSIDSBBCS.

Changing the CCSID Attribute of an Existing Column

If you want to change the CCSID attribute of an existing column, use the DBS utility. For example, to change the default CCSID for data in columns that were created previous to the migration to Version 3 Release 3, use the DBS utility to do the following:

1. Unload the data from the existing table.
2. Drop the table.
3. Recreate the table, specifying the new CCSID attribute for the column or columns that you want to change, or use the default if it is appropriate.
4. Reload the data.

Note: You must use the DBSU "DATALOAD/DATAUNLOAD" commands, **NOT** the "UNLOAD/RELOAD" commands.

Changing the Subtype Attribute of an Existing Column

The subtype attribute is only used when the CCSID attribute is null. If you have migrated from a release previous to Version 3 Release 3, existing character columns will have a CCSID value of null. For these columns, the subtype value is used to indicate their CCSID value. The CCSID value is either the value for MCCSIDSBBCS (for a subtype of "S") or the value for MCCSIDMIXED (for a subtype of "M").

In some cases, columns with a null CCSID could have a subtype of "S" and contain mixed data. This can occur if the column was created without specifying the FOR MIXED DATA clause. In this case, the subtype must be changed to "M" in order for the correct CCSID to be used for this column. Otherwise, conversion errors can occur (for example, SQLCODE -330, SQLSTATE 22517).

To change the subtype, DBA authority is required to update the SYSTEM.SYSCOLUMNS table. Change the value in the SUBTYPE column from "S" to "M" for the required character column.

Setting the Application Requester Default CHARNAME and CCSIDs

The application requester default CHARNAME can be set to either for all application requesters, or it can be set for an individual application requester. For more information, see either “Setting the Default CHARNAME and CCSIDs for All Application Requesters” or “Setting the Default CHARNAME and CCSIDs for an Application Requester” on page 342.

For a discussion of how your choice of default CHARNAME and CCSID affects the performance of your system and the characters displayed, refer to “Determining CCSID Values” on page 337.

If you want to check the CHARNAME value, use SQLINIT QRY from an application requester. The value displayed is obtained from the file LASTING GLOBALV. The SQLINIT EXEC uses the following hierarchy to set the CHARNAME value in LASTING GLOBALV to:

1. The value supplied as a parameter to the SQLINIT EXEC (see “Setting the Default CHARNAME and CCSIDs for an Application Requester” on page 342).
2. The current value of CHARNAME in LASTING GLOBALV.
3. The value in the file SQLGLOB DEFAULT Q set by the SQLGLOB EXEC (see “Setting the Default CHARNAME and CCSIDs for All Application Requesters”).
4. The default value coded in the SQLINIT EXEC.

Note: If the PROTOCOL parameter is SQLDS for either the application server or the application requester, the application server ignores the CHARNAME value set for the application requester. In this case, the application server assumes that the application requester default CHARNAME is the same value as the application server default CHARNAME.

The only exception to this is the folding performed by the application requester. On the application requester, folding is performed based on the default CHARNAME, regardless of the PROTOCOL parameter specified. However, if the application server CHARNAME and the application requester CHARNAME are *not* the same, unexpected results can occur.

Setting the Default CHARNAME and CCSIDs for All Application Requesters

When you run the SQLGLOB EXEC on a database machine, the EXEC sets the application requester default initialization values, including the default for CHARNAME, for all application requesters that are linked to the production minidisk. You must have write access to the production minidisk to run the SQLGLOB EXEC.

To specify an SBCS CHARNAME as the default for the application requesters, run the SQLGLOB EXEC and specify an SBCS CHARNAME parameter value. For example:

```
SQLGLOB CHARNAME (ENGLISH)
```

To specify a mixed CHARNAME as the default for the application requesters, run the SQLGLOB EXEC and specify a mixed CHARNAME parameter value. For example:

```
SQLGLOB CHARNAME (KOREAN)
```

Attention: The SQLGLOB EXEC defaults do not apply to application requesters that have already run the SQLINIT EXEC (and thereby have values in their LASTING GLOBALV file).

For more information about the SQLGLOB EXEC, see the *DB2 Server for VM Database Administration* manual.

Setting the Default CHARNAME and CCSIDs for an Application Requester

To use a CHARNAME different from that specified by the SQLGLOB EXEC, run the SQLINIT EXEC on the application requester and specify the CHARNAME parameter. The value specified for CHARNAME will override the database-wide default established by running the SQLGLOB EXEC.

To use an SBCS CHARNAME as the default, run the SQLINIT EXEC on the application requester and specify an SBCS parameter value. For example:

```
SQLINIT CHARNAME (ENGLISH)
```

If you specify ENGLISH, the value for CCSIDSBCS is set to 37, and the value for both CCSIDGRAPHIC and CCSIDMIXED is set to 0.

To use a mixed CHARNAME as the default, run the SQLINIT EXEC on the application requester and specify a mixed parameter value. For example:

```
SQLINIT CHARNAME (KOREAN)
```

If you specify KOREAN, the value for CCSIDMIXED is set to 933, the value for CCSIDSBCS is set to 833, and the value for CCSIDGRAPHIC is set to 834.

Note: When running in single user mode, the application server acts as an application requester. Because of this, you must run the SQLINIT EXEC also at the application server.

For more information about the SQLINIT EXEC, see the *DB2 Server for VM Database Administration* manual.

Setting the Application Server Default Character Subtype

To set the application server default character subtype, you must update a row in the SYSTEM.SYSOPTIONS catalog table. You must have DBA authority to do so. The CHARSUB option specifies the default subtype for a column when SUBTYPE clause or the CCSID is not specified explicitly (for example, on a CREATE TABLE or ALTER TABLE statement).

Note: The character subtype is defined for the application server only. It is not defined for the application requester. The CREATE PACKAGE CHARSUB option or the preprocessor CHARSUB option defines the default subtype for a package. For more information on this option, see the *DB2 Server for VM Application Programming* or the *DB2 Server for VSE & VM SQL Reference* manuals.

The initial setting of the application server default character subtype is SBCS. To set it to mixed, issue:


```
UPDATE SYSTEM.SYSOPTIONS
SET VALUE = 'MIXED'
WHERE SQLOPTION = 'CHARSUB'
```

To reset the application server default character subtype to SBCS, issue:

```
UPDATE SYSTEM.SYSOPTIONS
SET VALUE = 'SBCS'
WHERE SQLOPTION = 'CHARSUB'
```

In both situations, the new setting does not become effective immediately. The new setting is not in effect until the next time the application server is started.

If anything other than 'MIXED' or 'SBCS' is specified for the application server default character subtype in the SYSOPTIONS table, SBCS is assumed and an error message is issued when the application server is started.

The application server default character subtype can only be mixed when the application server default CHARNAME is mixed. The application server default character subtype is forced to be SBCS when the application server default CHARNAME is an SBCS CHARNAME.

Setting the DBCS Option for the Application Server

The DBCS option for the application server is set by updating a field in the SYSTEM.SYSOPTIONS catalog table. You must have DBA authority to update the catalog table.

The initial setting of the DBCS option is NO. To set the DBCS option to YES, issue:

```
UPDATE SYSTEM.SYSOPTIONS
SET VALUE = 'YES'
WHERE SQLOPTION = 'DBCS'
```

To reset the DBCS option to NO, issue:

```
UPDATE SYSTEM.SYSOPTIONS
SET VALUE = 'NO'
WHERE SQLOPTION = 'DBCS'
```

In both situations, the new setting does not become effective immediately. The new setting is not in effect until the next time the application server is started.

If you specify anything other than YES or NO for the DBCS option, NO is assumed and an error message is issued during startup.

For more information, see “Using Double-Byte Character Set (DBCS)” on page 331.

Setting DBCS Option for Application Requestors

The DBCS option can be set either for all application requestors, or it can be set by an individual application requester. For more information see either “Setting the DBCS Option for all Application Requestors” on page 344, or “Setting the DBCS Option for an Application Requester” on page 344.

If you want to check the DBCS option, use SQLINIT QRY from an application requester.

Setting the DBCS Option for all Application Requesters

When you run the SQLGLOB EXEC on a database machine, the EXEC sets the DBCS option for all application requesters that are linked to the production minidisk. You must have write access to the production minidisk to run the SQLGLOB EXEC.

To specify that the DBCS option is set to YES as the default for the application requesters, run the SQLGLOB EXEC as follows:

```
SQLGLOB DBCS (YES)
```

To specify that the DBCS option is set to NO as the default for the application requesters, run the SQLGLOB EXEC as follows:

```
SQLGLOB DBCS (NO)
```

The SQLGLOB EXEC defaults do not apply to application requesters that have already run the SQLINIT EXEC (and thereby have values in their LASTING GLOBALV file).

For more information, see "Using Double-Byte Character Set (DBCS)" on page 331. For more information about the SQLGLOB EXEC, see the *DB2 Server for VM Database Administration* manual.

Setting the DBCS Option for an Application Requester

To use a DBCS option different from that specified by SQLGLOB, run the SQLINIT EXEC on the application requester and specify the DBCS parameter. The value specified for the DBCS option will override the database-wide default established by the SQLGLOB EXEC.

To set the DBCS option to YES, run the SQLINIT EXEC as follows:

```
SQLINIT DBCS (YES)
```

To set the DBCS option to NO, run the SQLINIT EXEC as follows:

```
SQLINIT DBCS (NO)
```

For more information about the SQLINIT EXEC, see the *DB2 Server for VM Database Administration* manual.

EUC Conversions

Extended UNIX Code (EUC) allows for a form of ASCII mixed data. It is an encoding scheme supported by UNIX in far eastern countries which allows for MBCS characters. Each EUC codepage is made up of three character sets, or planes, denoted by G0, G1, and G2 or four character sets, denoted by G0, G1, G2 and G3. The group in which the data belongs is determined by the range of its first and second bytes. G0 is comprised of single-byte characters and is the ASCII invariant coded character set. G1 characters are double-byte characters within another range. G2 and G3 characters are triple-byte characters, distinguished by the first byte and the range of the last three bytes.

EUC conversion is supported by the database manager. EUC characters are converted to SBCS or DBCS characters, or both.

Examples of Setting Values for an Installation

This section discusses two examples of using the application server default CHARNAME JAPANESE-ENGLISH (CCSID=5035). The first example shows how to specify this CHARNAME and enable mixed string manipulation. The second example shows how to specify this CHARNAME without enabling mixed string manipulation and how to prevent the verification of character strings that contain mixed data. (Mixed string manipulation is the ability to specify mixed SQL identifiers, such as columns.)

Example 1

Suppose that you want to use the mixed JAPANESE-ENGLISH CCSID, 5035, as your application server default CCSID, and you also want to have the ability to do mixed string manipulation. To do this, set up your environment as follows:

1. Ensure that your terminal controllers are generated to use the correct code pages.

The CCSID you want to use is 5035. You must define the controller to use the character set 1172 for the SBCS character set, and code page 1027 for the SBCS code page. For the DBCS characters, specify the character set 370 and the code page 300.

2. Install the database manager.

The application server default CCSID for a newly installed database manager is 500 (CHARNAME=INTERNATIONAL). After installation, the SYSTEM.SYSOPTIONS catalog table contains the following information:

```
CHARNAME=INTERNATIONAL (the name of 500)
CCSIDSBBCS=500
CCSIDMIXED=0
CCSIDGRAPHIC=0
DBCS=NO
CHARSUB=SBCS
.....
```

3. Change the value of the application server default CHARNAME to JAPANESE-ENGLISH

Start the application server by using the SQLSTART EXEC. Specify CHARNAME=JAPANESE-ENGLISH. Message ARI0159D is displayed that informs you that the new CHARNAME (JAPANESE-ENGLISH) is different from the current default (INTERNATIONAL). You are prompted to enter either 1 (YES) to change the default, 0 (NO) to leave the default unchanged, or 111 (QUIT) to shut down the application server. Type 1 (for YES) and press ENTER.

After the application server is started, the SYSTEM.SYSOPTIONS catalog table should contain the following information:

```

CHARNAME=JAPANESE-ENGLISH
CCSIDSBBCS=1027      (the single-byte portion of 5035)
CCSIDMIXED=5035
CCSIDGRAPHIC=4396   (the double-byte portion of 5035)
DBCS=NO
CHARSUB=SBCS
.....

```

4. To enable mixed string manipulation, change the value for DBCS in SYSTEM.SYSOPTIONS from NO to YES. You can use either ISQL or the DBS utility.
5. Because most of the character columns will contain mixed data, you should also change the value for CHARSUB from SBCS to MIXED.
6. To cause the DBCS and CHARSUB values in SYSTEM.SYSOPTIONS to be used as the new application server defaults, you must stop the application server, and then restart it.

The changes are now complete. The SYSTEM.SYSOPTIONS catalog table contains the following information:

```

CHARNAME=JAPANESE-ENGLISH
CCSIDSBBCS=1027      (the single-byte portion of 5035)
CCSIDMIXED=5035
CCSIDGRAPHIC=4396   (the double-byte portion of 5035)
DBCS=YES
CHARSUB=MIXED
.....

```

7. To set these values for all the application requesters, run the SQLGLOB EXEC. Issue the following command:

```
SQLGLOB CHARNAME(JAPANESE-ENGLISH) DBCS(YES)...
```

Note that CHARSUB is only applicable to the application server.

Example 2

Suppose that you want to use the mixed JAPANESE-ENGLISH CCSID, 5035, as your application server default CCSID. Because you must be able to both store DBCS characters, and retrieve DBCS characters from graphic columns (GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC), you cannot specify an ENGLISH single-byte CCSID such as 37 or 1027. Also suppose that you do not want the ability to do mixed string manipulation, and you want to prevent the database manager from verifying character strings for mixed data. In addition, you also want character columns that are created without the explicit specification of a CCSID or a subtype to default to the SBCS subtype and CCSID. To do this, set up your environment as follows:

1. Ensure that your terminal controllers are generated to use the correct code pages.

The CCSID you want to use is 5035. You must define the controller to use the character set 1172 for the SBCS character set, and code page 1027 for the SBCS code page. For the DBCS characters, specify the character set 370 and the code page 300.

2. Install the database manager.

The application server default CCSID for a newly installed database manager is 500 (CHARNAME=INTERNATIONAL). After installation, the SYSTEM.SYSOPTIONS catalog table contains the following information:

```
CHARNAME=INTERNATIONAL (the name of 500)
CCSIDSBBCS=500
CCSIDMIXED=0
CCSIDGRAPHIC=0
DBCS=NO
CHARSUB=SBCS
.....
```

3. Change the value of the application server default CHARNAME to JAPANESE-ENGLISH.

Start the application server by using the SQLSTART EXEC. Specify CHARNAME=JAPANESE-ENGLISH. Message ARI0159D is displayed that informs you that the new CHARNAME (JAPANESE-ENGLISH) is different from the current default (INTERNATIONAL). You are prompted to enter either 1 (YES) to change the default, 0 (NO) to leave the default unchanged, or 111 (QUIT) to shut down the application server. Type 1 (for YES) and press ENTER.

After the application server is started, the SYSTEM.SYSOPTIONS catalog table should contain the following information:

```
CHARNAME=JAPANESE-ENGLISH
CCSIDSBBCS=1027      (the single-byte portion of 5035)
CCSIDMIXED=5035
CCSIDGRAPHIC=4396  (the double-byte portion of 5035)
DBCS=NO
CHARSUB=SBCS
.....
```

4. Because you do not want to enable mixed string manipulation, and you do not want the database manager to verify character strings for mixed data, leave the DBCS option set to NO (even though the database manager uses a mixed CCSID). This still allows you to:
 - Issue CREATE TABLE or ALTER TABLE statements to either add or create GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC columns. The CCSID for these columns will be taken from value for CCSIDGRAPHIC in the SYSTEM.SYSOPTIONS catalog table.
 - Insert into graphic columns from graphic host variables.
 - Select graphic columns into graphic host variables.
 - Use graphic constants in SQL statements.
5. Because most character columns will contain SBCS data, leave the value for CHARSUB as SBCS. When you need to either create or add a mixed character column, you can specify the FOR MIXED DATA clause or the CCSID clause explicitly for the CREATE TABLE or the ALTER TABLE statement.
6. To set these values for all the application requesters, run the SQLGLOB EXEC. Issue the following command:

```
SQLGLOB CHARNAME(JAPANESE-ENGLISH) DBCS(YES)...
```

For an application requester to be able to use graphic data, the application requester must use a mixed CCSID as the default. One exception exists. If

SQLINIT PROTOCOL(SQLDS) is issued from an application requester (or if the application server was started with SQLSTART PROTOCOL(SQLDS)), the application server responds to the application requester with the expectation that the application requester is using the same mixed CCSID as the application server is using. If the user specified a different value for the SQLINIT CHARNAME parameter, the application server ignores this value. (However, folding performed by the application requester is still based on the application requester default CHARNAME, regardless of the PROTOCOL parameter specified. In this case, if the application server CHARNAME and the application requester CHARNAME are *not* the same, unexpected results can occur. See “Setting the Application Requester Default CHARNAME and CCSIDs” on page 341 for more information.)

In this step, the DBCS option is set to YES. Because DBCS strings can be truncated when displayed by ISQL, setting the DBCS option to YES ensures that truncation is done properly. If truncation is not a consideration, you can set the DBCS option to NO.

Identifying Classification and Translation Tables for a CCSID

To identify either the classification table or the translation table that is used for folding characters to uppercase for a specific CCSID, do the following:

1. Review the CHARNAME column of the SYSTEM.SYSCCSIDS catalog table for the CHARNAME value of the CCSID.
2. Review the NAME column of the SYSTEM.SYSCCHARSETS catalog table for the value that matches the CHARNAME of the CCSID. That row contains both the classification table and the translation table for the CCSID.

National Language Support for Messages and HELP Text

The database manager can provide DB2 Server for VM messages and HELP text in several national languages. Messages and HELP text come with the product tape. For more information on HELP text, see the *DB2 Server for VM Database Administration* manual.

When the national language feature tape has been installed, national language support works this way:

- ISQL users can receive messages in the language they select.
- Users of DB2 Server for VM EXECs receive messages in the language that is the current language setting in CMS.
- Users of the DBS utility and the preprocessors receive messages in the language that is the current language setting in CMS.
- ISQL users can receive HELP text for commands and messages in the language they select.
- The DB2 Server for VM operator can receive messages on the operator console in the language selected.

The national language tape provided contains the following languages:

- Mixed American English
- Uppercase American English
- French

- German
- Japanese
- Simplified Chinese.

When the database manager is installed, you specify a default national language. This is a second-level default. The first-level default is the current CMS language setting. If that language is unavailable in the database manager, the second-level default is used. When one or more additional national languages have been installed, users can change the language from the default in the following ways:

- ISQL users can choose the language for messages and HELP text using the SET LANGUAGE command. For ISQL users to receive HELP text in the language they choose, the messages and the HELP text for that language must be installed. To support a national language, you must install the messages for that language. Installing the HELP text is optional.

If the language is installed on the database manager and in CMS, the user can issue SET LANGUAGE for that language in CMS. If the language is installed on the database manager but is not supported in CMS, the user cannot issue SET LANGUAGE for that language in CMS. When ISQL is being started, the user will receive messages in the current language setting in CMS (if the current language setting in CMS is unavailable in the database manager, the DB2 Server for VM default language is used). After ISQL has been started, the user can change the language for messages and HELP text.

The following paragraph only applies if you have VSE guest sharing:

The VSE online users can also choose the language they receive messages in by specifying the LANGID parameter on the CIRB transaction. For an explanation of the CIRB transaction, see “Starting the Online Resource Adapter -- The CIRB Transaction” on page 106.

- Users of the DBS utility, DB2 Server for VM EXECs, and the preprocessors can only receive messages in national languages supported by the operating system. They select a language by issuing SET LANGUAGE in CMS. If the current CMS language setting is not installed on the database manager, the default national language is used if it is supported by the operating system.
- DBS utility users and preprocessor users on a VSE guest receive messages in the language specified when the database manager was installed. If a user wants to receive messages in another language, the user should specify the library containing the desired language in the LIBDEF statement of the job control.
- The operator can choose the language for operator messages using the SET LANGUAGE command from the operator console. This language is also used to display the output of the SHOW, RESET, and COUNTER commands.

National languages are identified to the database manager by a language name, and a LANGID (language identifier). These values are in the SQLDBA.SYSLANGUAGE table. If you have English and French installed on the database manager, the SQLDBA.SYSLANGUAGE table can look like the example in Figure 107 on page 350.

LANGUAGE	LANGKEY	REMARKS	LANGID
ENGLISH	S001	ENGLISH VERSION OF HELP TEXT	AMENG
FRANCAIS	S003	TEXTE D'AIDE FRANCAIS	FRANC

Figure 107. Sample SQLDBA.SYSLANGUAGE Table

For the LANGUAGE and REMARKS columns, you can choose values appropriate for your organization. For the LANGKEY and LANGID columns, you should keep the values supplied by the database manager.

The language keys (LANGKEY) and language identifiers (LANGID) used by the database manager are shown in Table 24.

LANGUAGE	LANGKEY	LANGID
ENGLISH (mixed case)	S001	AMENG
ENGLISH (uppercase)	S002	UCENG
FRENCH	S003	FRANC
GERMAN	S004	GER
JAPANESE	D001	KANJI
CHINESE_SIMPLIFIED	D003	HANZI

You should not use the language keys and language identifiers (LANGID) shown above for other purposes. In addition, the language keys S007-S500 and D003-D500 are reserved for IBM use.

The language key is used to internally identify HELP text for a language. The LANGID can be used to choose a language for messages and HELP text. You can also specify the name of the language, as it is stored in the LANGUAGE column of the SQLDBA.SYSLANGUAGE table.

In ISQL, and on the operator console, you can specify a language or a LANGID on the SET LANGUAGE command. The syntax of the SET LANGUAGE command is shown in Figure 108.

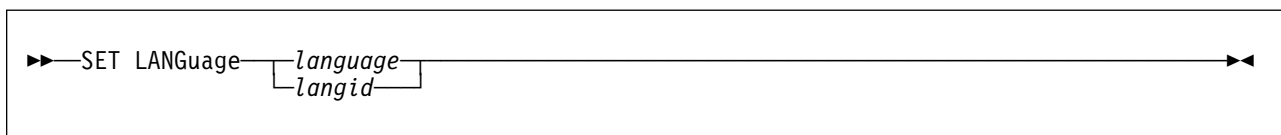


Figure 108. The SET LANGUAGE Command

The language or LANGID you specify must match a value in the SYSLANGUAGE table, and must be installed. If your installation uses a double-byte character set, you should consider setting the DBCS option to YES. For information on the DBCS option, see “Using Double-Byte Character Set (DBCS)” on page 331.

When using the LANGID parameter on the CIRB transaction, (if you have VSE guest sharing), or in CMS, you can specify only the LANGID. The LANGIDs used by the database manager are consistent with CMS LANGIDs. When a national language is not supported by CMS, but it is supported by the database manager,

the database manager uses its own LANGID for the language. In ISQL, on the operator console, or when using the CIRB transaction (for guest sharing), you can use any LANGID shown in Table 24 if the language is installed on the database manager.

Note: Not all LANGIDs are supported in all operating systems.

CMS HELP Text Files

When you install the database manager, you are prompted for a location in which to install the CMS HELP text files. If you specify the production minidisk address (195), CMS HELP text in languages other than American English will be unavailable. If you install messages in more than one national language, you need a separate minidisk for each national language. Ensure that the minidisk (or minidisks) used cannot be accessed by other user IDs, including the database machine.

When installing the CMS Help files, you must use the disk known by the VM system for that language. The address of this disk is identified by the LANGMERC control file. For additional information on the LANGMERC control file, see the *VM/ESA: CMS Command Reference* manual.

National Language Messages in a VSE Guest Sharing Environment

If you have VSE guest sharing, you should install all languages on the VSE guest that you want to support on VSE. Users who use the DBS utility and the preprocessors from the VSE guest should specify the library containing the desired language in the LIBDEF statement of the job control.

Defining Message Repositories as Saved Segments

The DB2 Server for VM messages for each national language you have installed are stored in message repositories. There are two message repositories for each language. One (ARIMxxxx where xxxx represents the LANGKEY) is for messages that are issued while the database is running. The other (ARIUME) is installed according to the CMS-specified rules for NLS (national language support) support for CMS applications, and uses the CMS facility to issue NLS messages from EXECs. The ARIUME message repository is used during initialization or termination of services (that is by the database manager itself, ISQL, DBS utility, and preprocessors among other things) since the message services cannot be used at that time.

For each language that is frequently used, you can have ARIMxxxx and ARIUME as follows:

- ARIMxxxx

The ARIMxxxx repository (where xxxx represents the LANGKEY) is the repository for DB2 Server for VM messages, and is used by modules that reside in the user machine. (The ARIMxxxx repository is also used by the modules in the database machine. However, the database machine will always load ARIMxxxx into free storage). The saved segment is identified to the database manager by the ARISNLSC MACRO. Defining ARIMxxxx as a saved segment is described in “Defining Saved Segments” on page 187. To refresh this segment after service, you have to run ARISDBMA.

- ARIUME

The DB2 Server for VM EXEC repository (ARIUME) is accessed by the database and user machines, and uses the CMS facility to issue NLS messages from EXECs. It is installed according to the rules specified by CMS for NLS support for CMS applications, and is added to the segment that contains the CMS message repository. The CMS message repository for the default language can be defined as a nucleus extension to which the ARIUME message repository can be added. For a language other than the default you must first define the segment for that language. For more information, see the *VM/ESA: Installation Guide*. The name of this segment will be NLXY, where *X* is the *langlev* specified in DMSNGP ASSEMBLE LANGLEV parameter, and *Y* is the *LANGID* as specified in Table 24 on page 350. For example, if you use the default LANGLEV=S, and are installing American English (AMENG), then the segment name would be NLSAMENG.

To add DB2 Server for VM messages to the nucleus extension containing the CMS messages (for the default language) or to the saved segment containing CMS messages (for non-default languages), perform the following steps:

1. Define an NLSlangid segment, if one had not been defined yet, using the instructions in the "Installing and Customizing National Language" chapter in the *VM/ESA: Installation Guide*.
2. Make sure that the ARIUME repository is placed on a file mode accessible by the machine (for example, file mode A).
3. Update the LANGMERC control file.

You need to have a LANGMERC control file for the national language you are saving as a saved segment. See the LANGMERC command in the *VM/ESA: CMS Command Reference* manual. The line added for the database manager should look as follows:

```
MESSAGE ARIUME TXTlangid *
```

If this file does not exist in the system, create this file and add the above line into the file. The *langid* stands for the language identifier of the national language you are defining the saved segment for.

If the language is AMENG and the SQL message repository is named ARIUME TEXT, this line should look as follows:

```
MESSAGE ARIUME TEXT *
```

4. Issue the LANGMERC command.

The syntax of the command is:

```
LANGMERC langid ARI ( CTL fn
```

The *langid* stands for the language identifier of the national language, and *fn* is the file name of the control file created or updated in step 3.

For example, if the language is AMENG and the control file is DMSNLS LANGMCTL, the command should look as follows:

```
LANGMERC AMENG ARI ( CTL DMSNLS
```

This will create a file ARINLS TXTAMENG on your A disk.

5. Update the LANGGEN control file.

You have to add a line to the LANGGEN control file for the language identifier for the national language you are saving as a saved segment. See

the LANGGEN command in the *VM/ESA: CMS Command Reference* manual. The line should look as follows:

```
ARINLS TXTlangid A
```

The *langid* stands for the language identifier of the national language.

If the LANGGEN control file does not exist in the system, create one and add the CMS message repository file in front of the SQL message repository file. The lines should look as follows:

```
DMSMES TEXT *  
ARINLS TXTlangid A
```

DMSMES TEXT is the name of the English CMS message repository file. Use the correct message repository file name for the national language chosen (see Step 4 on page 352).

6. Issue the LANGGEN command.

The syntax of the command is:

```
LANGGEN langid ( CTL fn
```

The *langid* stands for the language identifier of the national language, and *fn* is the file name of the control file created or updated in step 3 on page 352.

For example, if the language is AMENG and the control file is DMSNLS LANGGCTL, the command should look as follows:

```
LANGGEN AMENG ( CTL DMSNLS
```

LANGEN will invoke SAVESEG or SAVESYS to save the NLS language segment.

7. Build and load the segment using the instructions in the “Installing and Customizing National Language” chapter in the *VM/ESA: Installation Guide*.

8. Issue the SET LANGUAGE command from each machine that you want to be able to access the saved segment.

For example:

```
SET LANGUAGE AMENG ( ADD ARI SYSTEM
```

This command can be added to the PROFILE EXEC, or issued as a CMS command before running or accessing the database manager.

After applying service to module ARIUME, you will need to repeat step 6 and step 7.

For complete instructions on how to set up an application message repository, see the national language information supplied with VM/ESA.

Chapter 14. Creating Installation Exits

This chapter discusses installation exits that:

- Supply account numbers for product users
- Define your own datetime format
- Coding your own TRANSPROC exit
- Perform your own cancel exit
- Encode and decode data (Field Procedures).

Supplying Account Numbers for Users

There is no rigid format for entering account or project numbers into accounting records, because their definition and use vary at each installation. (Some installations do not use account numbers at all.) Thus, you must devise your own scheme.

To do this, you replace a module named ARIUXIT with your own version of that module.

Note: The ARIUXIT accounting exit is not called in a DRDA protocol environment.

The resource adapter calls ARIUXIT when a user tries to connect to a DB2 Server for VM application server either implicitly or explicitly. The database manager branches to ARIUXIT even before attempting to verify the user.

The database manager allows ARIUXIT to access a control block. In this control block, ARIUXIT can provide up to 16 bytes of data.

Before calling ARIUXIT, the database manager initializes the 16-byte area. For batch/ICCF applications, the database manager initializes the area to character blanks.

The ARIUXIT module does not use the control block (except for the return code area); it only sets a no-operation return code and branches back to the database manager, as shown in the following figure.

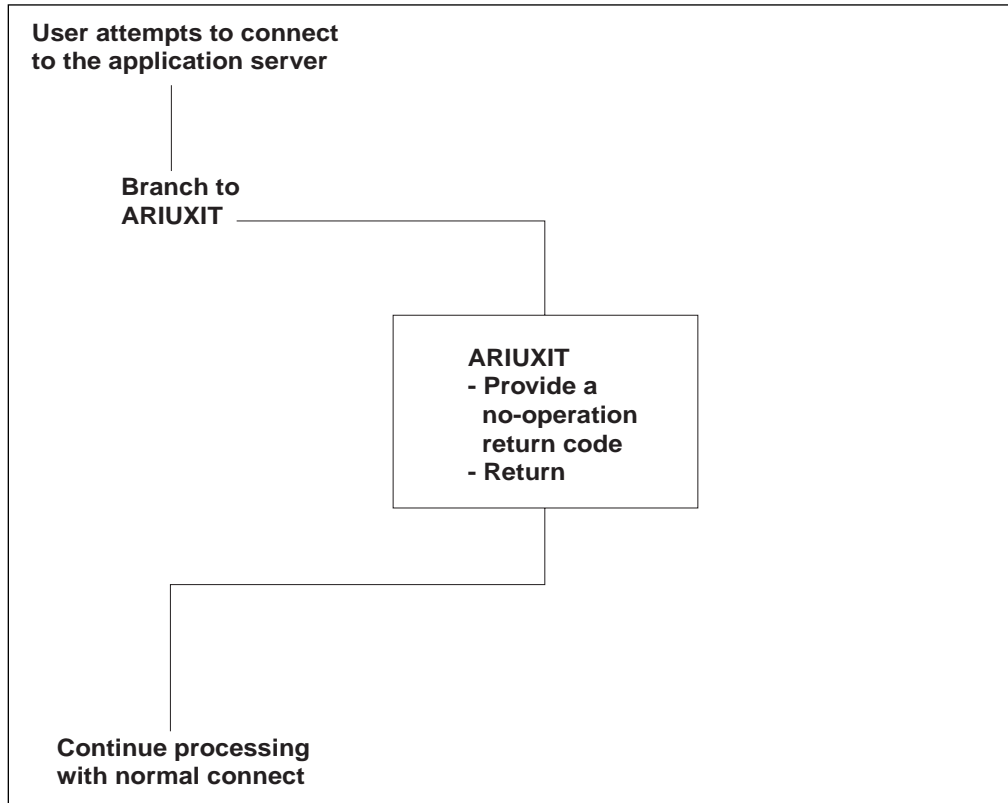


Figure 109. The Database Manager Branching to ARIUXIT

Because of this, the database manager places blanks in the installation-dependent field of the user's accounting records. Whatever data is in the 16-byte area is placed on the accounting records of the user who was trying to connect at the time that ARIUXIT was called.

Your version of ARIUXIT should determine the user's accounting information for your installation, verify it, and pass it to database manager which puts it in the user's accounting records. You can supply department names as well as account or project numbers. You can, in fact, supply whatever you like so long as it fits in 16 bytes and meets your own installation's requirements. The database manager does no error-checking on the data.

The database manager **always** branches to ARIUXIT in a non-DRDA protocol environment, regardless of whether the connect attempt is from a program, the DBS utility, the preprocessors, or ISQL. You cannot disable branching. If you want to be able to bypass your accounting routine, you have to code the routine so that you can turn it on and off.

How the ARIUXIT Module Works

The resource adapter is the component of the database manager that calls ARIUXIT. In multiple user mode, the resource adapter is in the user machine; in single user mode, it is in the database machine. The ARIUXIT module is called in both modes.

The resource adapter calls ARIUXIT in AMODE(31).

When the resource adapter detects any attempt to connect to an application server, it builds a parameter list for ARIUXIT, sets registers for a proper linkage, and calls ARIUXIT. It always calls ARIUXIT, even if the accounting facility is not enabled. The registers are set as follows:

Register 1 The address of the start of the parameter list for ARIUXIT. The parameter list itself is named ARIUEXI. The pointer to the parameter list points to the beginning of ARIUEXI, which is described below. The first field in ARIUEXI is an eye-catcher value.

Register 13 Points to a standard register save area.

Register 14 Contains the return address.

Register 15 Contains the entry point of the installation exit routine.

You must code ARIUXIT to save the DB2 Server for VM registers in the area pointed to by Register 13. If ARIUXIT does not save and restore the registers, the results will be unpredictable.

The resource adapter also builds the parameter list named ARIUEXI. Table 25 shows what is in ARIUEXI.

Length	Description
2 words 1 word	Eye-catcher: 'ARIUEXI ' Length of ARIUEXI parameter list
1 word 1 word	Pointer to Exit Number Pointer to length of Exit Number
1 word 1 word	Pointer to Exit Global Area Pointer to length of Exit Global Area
1 word 1 word	Pointer to Exit Local User Area Pointer to length of Exit Local User Area
1 word 1 word	Pointer to Exit Unique Area Pointer to length of Exit Unique Area
2 words	Reserved
1 word 1 word	Pointer to Environment Dependent Area Pointer to length of Environment Dependent Area
1 word 1 word	Pointer to Exit Return Code Area Pointer to length of Exit Return Code Area

Each area that ARIUEXI points to is described below.

Eye-catcher and Length of List

The resource adapter sets the eye-catcher field to 'ARIUEXI ' and the following full word to the length of the entire list. (This length includes the length of the eye-catcher field.)

Exit Number

The exit number is always a full word. The exit number for the accounting exit is 1. The resource adapter sets the pointer to the exit number with an address to a full word area containing a binary 1. The resource adapter sets the pointer to the length of the exit number with the address of a full word area containing a binary 4.

Exit Global Area

This area does not apply to the accounting exit. The resource adapter sets both the pointer to the global area and the pointer to the length of the global area to binary zeros.

Exit Local User Area

The local user area is 16 bytes long. It is a read/write area that lasts for the life of the user program. For CMS applications, the area exists until the end of the CMS command.

For each user, the resource adapter obtains the 16-byte storage area and sets it to binary zeros. The pointer to the local user area is unique for each user. On subsequent calls by the user, the resource adapter returns the same pointer; it never resets the area.

The pointer to the length of the local user area always points to a full word that contains a binary 16.

Exit Unique Area

In this area, you provide the installation-dependent accounting information. This area is also 16 bytes long. The resource adapter initializes the field to character blanks. In the VM/ESA operating system, the first four bytes of this area contain the CMS work unit ID. The accounting exit writes over these first four bytes unless you modify it.

The pointer to the length of the area points to a full word that contains binary 16.

Reserved Area

This area is 8 bytes long and reserved. The resource adapter initializes it to binary zeros.

Environment-Dependent Area

This area is 40 bytes long. It contains information about the environment where the user is running.

Note: Some fields apply only to VSE uses of the database manager. For VM, those fields are set to binary zeros or character blanks.

The resource adapter initializes the environment-dependent area as follows:

Byte 1 Character S for single user mode, or M for multiple user mode.

Byte 2 Character V for VM.

Byte 3 Character blank for VM.

Byte 4 Character I for implicit connect, or E for explicit connect.

Bytes 5—8
Binary zeros.

Bytes 9—12
Binary zeros.

Bytes 13—20
Database name.

Bytes 21—28
CONNECT user ID for all explicit connections. Blanks for implicit connections.

Bytes 29—36

Character blanks.

Bytes 37—40

Binary zeros (reserved).

For implicit connections, you can use the CP DIAGNOSE instruction X'00' to obtain the VM user ID. (The VM user ID is used as the DB2 Server for VM user ID for implicit connections.)

Exit Return Code Area

The resource adapter initializes this full word area (and the pointer to it), and sets it to -1. A return code of -1 means that you do not want this exit. The length field for this area is a full word containing a binary 4. The resource adapter also ORs a X'80' to the high order byte of the pointer to the length field of the return code area. The X'80' indicates the end of the parameter list.

When you code your version of ARIUXIT, you can specify these return codes before branching back to the database manager:

- 1** Means that you do not want this exit (the default). This indicates to the database manager that the exit is a no-op.
- 0** The function that the exit called to do a task completed successfully.
- Other** Any return code other than 0 or -1 causes an -815 SQLCODE to be returned to the user. (SQLERRD1 contains the return code from the exit.) You can reject a user's attempt to connect because the user has incorrect accounting information.

Figure 110 on page 360 summarizes the ARIUEXI parameter list and the areas pointed to by the list.

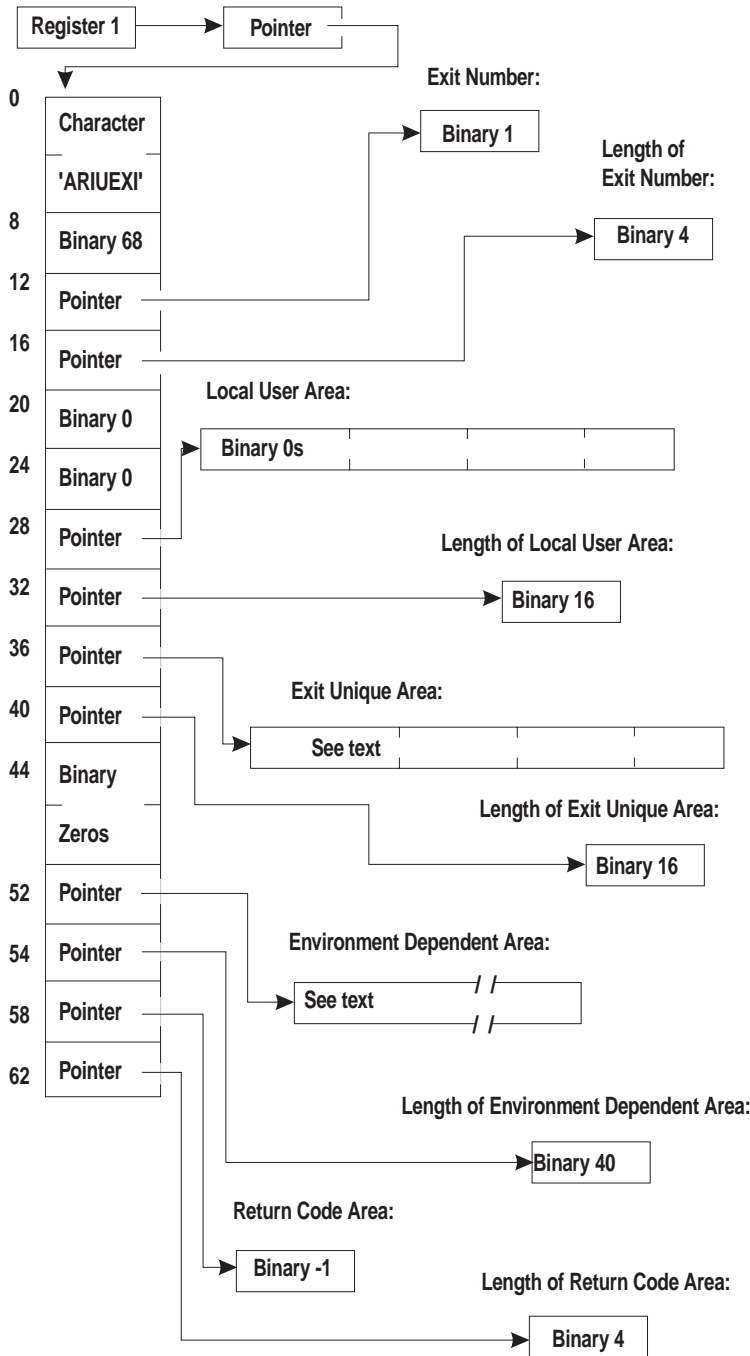


Figure 110. Summary of ARIUEXI Parameter List and Associated Areas

Coding Your Own Accounting Exit

Exit routines must always be coded in Assembler language. Your version of ARIUXIT (and any of the modules it calls) must not use any DB2 Server for VM function.

Figure 111 on page 361 shows the ARIUXIT module that is included with the database manager. This sample exit routine is on the service disk. Its file name and file type are ARIUXIT MACRO. Note that the Exit Return Code Area is set to -1, which means that you are not interested.

```

          TITLE 'ARIUXIT'
*****
* ARIUXIT USER EXIT ROUTER ROUTINE
* REGISTER ASSUMPTIONS:
*   R1  -> PARMLIST
*   R13 -> SAVE AREA
*   R14 -> RETURN ADDRESS
*   R15 -> ENTRY POINT
*
* ALTHOUGH PROVIDED IN A GENERAL INTERFACE LIBRARY, ARIUXIT IS NOT TO
* BE USED AS A GENERAL PROGRAMMING INTERFACE. REFER TO PRODUCT
* DOCUMENTATION TO DETERMINE INTENDED USAGE.
*
*****
          SPACE 5
ARIUXIT  CSECT ,
ARIUXIT  AMODE ANY
ARIUXIT  RMODE 24
          DS    0H
          USING *,R15          GET ADDRESSABILITY
          B     PROLOG
          DC    CL8'ARIUXIT '  EYECATCHER
*
PROLOG   EQU   *
          STM   R14,R12,12(R13)  SAVE CALLER'S REGISTERS
          DROP  R15
          BALR  R12,0            R12 IS BASE REGISTER
*
PSTART  EQU   *
          USING PSTART,R12      GET ADDRESSABILITY FOR ROUTINE
          ST    R13,UXSAVE+4    STORE BACKWARD POINTER
          LA   R9,UXSAVE        ADDRESS OF SAVE AREA
          ST   R9,UXSAVE+8      STORE FORWARD POINTER
          LR   R13,R9           R13 POINTS TO NEW SAVE AREA
          L    R1,0(,R1)        GET POINTER TO PLIST
          USING PLIST,R1        GET ADDRESSABILITY TO PLIST DSECT
*
* Insert your own code here
* (and change the return code as appropriate).
*

```

Figure 111 (Part 1 of 3). IBM-Supplied Version of ARIUXIT

```

        L    R2,PLRETC      GET PTR TO EXIT RETURN CODE AREA
        L    R3,NEG1RC     LOAD NOOP RET CODE (NEGATIVE ONE)
*
        ST   R3,0(,R2)     STORE RET CODE INTO EXIT RC AREA
*
        L    R13,UXSAVE+4  GET BACKWARD POINTER
        LM   R14,R12,12(R13) RESTORE CALLER'S REGISTERS
        BR   R14           RETURN TO CALLER
*
END     EQU   *
        EJECT
*****
*
*       DECLARES FOR ARIUXIT ROUTER
*
*****
        SPACE 5
UXSAVE  DC   18F'0'        SAVE AREA FOR CALLER'S REGISTERS
NEG1RC  DC   F'-1'        NEGATIVE ONE RETURN CODE (NO-OP)
        SPACE 2
R0      EQU   0           REGISTERS EQUATES
R1      EQU   1
R2      EQU   2
R3      EQU   3
R4      EQU   4
R5      EQU   5
R6      EQU   6
R7      EQU   7
R8      EQU   8
R9      EQU   9
R10     EQU   10
R11     EQU   11
R12     EQU   12
R13     EQU   13
R14     EQU   14
R15     EQU   15
        EJECT

```

Figure 111 (Part 2 of 3). IBM-Supplied Version of ARIUXIT

```

*****
*
*           DSECT FOR ARIUEXI PARAMETER LIST INTERFACE TO ARIUXIT ROUTER
*
*****
          SPACE 5
          DS    0D
PLIST    DSECT
PLICTCH DS   CL8           EYECATCHER
PLILENG DS    F           LENGTH OF PLIST (INCLUDING EYECATCHER)
PLEXNUM DS    F           PTR TO EXIT NUMBER
PLLXNUM DS    F           PTR TO LENGTH OF EXIT NUMBER
PLGLOBA DS    F           PTR TO EXIT GLOBAL AREA
PLLGLOB DS    F           PTR TO LENGTH OF EXIT GLOBAL AREA
PLUSERF DS    F           PTR TO EXIT LOCAL USER AREA
PLLUSER DS    F           PTR TO LENGTH OF EXIT LOCAL USER AREA
PLEUNIQ DS    F           PTR TO EXIT UNIQUE AREA
PLLUNIQ DS    F           PTR TO LENGTH OF EXIT UNIQUE AREA
          DS   CL8           RESERVED
PLEDEPA DS    F           PTR TO ENVIRONMENT DEPENDENT AREA
PLLDEPA DS    F           PTR TO LENGTH OF ENVIRONMENT DEP AREA
PLRETC  DS    F           PTR TO EXIT RETURN CODE AREA
PLLRETC DS    F           PTR TO LENGTH OF EXIT RETURN CODE AREA
*
          END

```

Figure 111 (Part 3 of 3). IBM-Supplied Version of ARIUXIT

Figure 112 shows a simple example of a user version of ARIUXIT. In this example, the string HERE IS USERDATA is moved into the exit unique area, and the exit return code area is set to 0.

```

          TITLE 'ARIUXIT'
*****
* ARIUXIT USER EXIT ROUTER ROUTINE
* REGISTER ASSUMPTIONS:
*   R1  -> PARMLIST
*   R13 -> SAVE AREA
*   R14 -> RETURN ADDRESS
*   R15 -> ENTRY POINT
*
* ALTHOUGH PROVIDED IN A GENERAL INTERFACE LIBRARY, ARIUXIT IS NOT TO
* BE USED AS A GENERAL PROGRAMMING INTERFACE. REFER TO PRODUCT
* DOCUMENTATION TO DETERMINE INTENDED USAGE.
*
*****
          SPACE 5
ARIUXIT  CSECT ,
ARIUXIT  AMODE ANY
ARIUXIT  RMODE 24
          DS    0H
          USING *,R15          GET ADDRESSABILITY
          B     PROLOG
          DC    CL8'ARIUXIT '  EYECATCHER
*
PROLOG   EQU   *
          STM   R14,R12,12(R13)  SAVE CALLER'S REGISTERS
          DROP  R15
          BALR  R12,0            R12 IS BASE REGISTER
*
PSTART  EQU   *
          USING PSTART,R12      GET ADDRESSABILITY FOR ROUTINE
          ST    R13,UXSAVE+4    STORE BACKWARD POINTER
          LA   R9,UXSAVE        ADDRESS OF SAVE AREA
          ST    R9,UXSAVE+8     STORE FORWARD POINTER
          LR   R13,R9           R13 POINTS TO NEW SAVE AREA
          L    R1,0(,R1)        GET POINTER TO PLIST
          USING PLIST,R1        GET ADDRESSABILITY TO PLIST DSECT

```

Figure 112 (Part 1 of 4). Sample User Version of ARIUXIT

```

*
* Here you would place code that gets and verifies your
* user-dependent data. The following code shows moving the data
* into the Exit Unique Area.
*
* Make sure you check the Exit Number word. If the Exit Number value
* is not a binary 1, you should set the Exit Return Code word to binary
* -1 (NEG1RC) and return to the database manager.
*
      L    R2,PLEUNIQ          GET PTR TO EXIT UNIQUE AREA
      MVC  0(16,R2),USERDATA   MOVE 16 BYTES OF USER DATA
      L    R2,PLRETCO          GET PTR TO EXIT RETURN CODE AREA
      L    R3,ZEROS            SET ZERO RETURN CODE
      ST   R3,0(,R2)           STORE RET CODE INTO EXIT RC AREA
*
EXIT   EQU  *                  RETURN TO THE DATABASE MANAGER
      L    R13,UXSAVE+4        GET BACKWARD POINTER
      LM   R14,R12,12(R13)     RESTORE CALLER'S REGISTERS
      BR   R14                  RETURN TO CALLER
*
END    EQU  *
      EJECT

```

Figure 112 (Part 2 of 4). Sample User Version of ARIUXIT

```

*****
*
*          DECLARES FOR ARIUXIT
*
*****
      SPACE 5
UXSAVE  DC  18F'0'            SAVE AREA FOR CALLER'S REGISTERS
ZEROS   DC  F'0'              ZERO RETURN CODE
NEG1RC  DC  F'-1'            NEGATIVE RETURN CODE (NO-OP)
USERDATA DC  CL16'HERE IS USERDATA'
      SPACE 2
R0      EQU  0                REGISTERS EQUATES
R1      EQU  1
R2      EQU  2
R3      EQU  3
R4      EQU  4
R5      EQU  5
R6      EQU  6
R7      EQU  7
R8      EQU  8
R9      EQU  9
R10     EQU  10
R11     EQU  11
R12     EQU  12
R13     EQU  13
R14     EQU  14
R15     EQU  15
      EJECT

```

Figure 112 (Part 3 of 4). Sample User Version of ARIUXIT

```

*****
*
*           DSECT FOR ARIUEXI PARAMETER LIST INTERFACE TO ARIUXIT
*
*****
          SPACE 5
          DS      0D
PLIST    DSECT
PLICTCH  DS      CL8           EYE-CATCHER
PLILENG  DS      F            LENGTH OF PLIST (INCLUDING EYE-CATCHER)
PLEXNUM  DS      F            PTR TO EXIT NUMBER
PLLXNUM  DS      F            PTR TO LENGTH OF EXIT NUMBER
PLGLOBA  DS      F            PTR TO EXIT GLOBAL AREA
PLLGLOB  DS      F            PTR TO LENGTH OF EXIT GLOBAL AREA
PLUSERF  DS      F            PTR TO EXIT LOCAL USER AREA
PLLUSER  DS      F            PTR TO LENGTH OF EXIT LOCAL USER AREA
PLEUNIQ  DS      F            PTR TO EXIT UNIQUE AREA
PLLUNIQ  DS      F            PTR TO LENGTH OF EXIT UNIQUE AREA
          DS      CL8           RESERVED
PLEDEPA  DS      F            PTR TO ENVIRONMENT DEPENDENT AREA
PLLDEPA  DS      F            PTR TO LENGTH OF ENVIRONMENT DEP AREA
PLRETC   DS      F            PTR TO EXIT RETURN CODE AREA
PLLRETC  DS      F            PTR TO LENGTH OF EXIT RETURN CODE AREA
*
          END

```

Figure 112 (Part 4 of 4). Sample User Version of ARIUXIT

After the program is coded, assemble it as you would any other program.

Installing Your Version of ARIUXIT

During customization of DB2 Server for VM, you may have to modify TEXT files that are serviced as full part replacement (for example, for user exits, such as ARIUXIT). The source is shipped as a MACRO file.

Use the following steps to create and build the affected objects with the new parts. These steps use the VMSES/E local modification procedure.

Step 1. Stop the Application Server

Stop the application server using your normal operating procedures.

Step 2. Log on to the Installation or Service User ID

Log on to the installation or service user ID, 5648A70S.

Step 3. Establish the Minidisk or SFS Directory Order

Establish the access order.

```
vmfsetup 5648A70S {DB2VM|DB2VMSFS}
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

If you have your own PPF override, then substitute that name for the 5648A70S name shown in this command and any following commands.

Step 4. Copy the MACRO Source Code (First Time Only)

Copy the MACRO source code to the local modification disk as ASSEMBLE. This only needs to be done the first time you apply local modifications to this part; otherwise, you will delete your version of the ASSEMBLE file.

```
vmfrep1 ariuxit macro 5648A70S {DB2VM|DB2VMSFS}
  (filetype assemble outmode localsam
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

Step 5. Edit the ASSEMBLE File

Edit ARIUXIT ASSEMBLE file on the local modification disk (2C2) and make your changes.

Step 6. Update the Local Version Vector Table

Update the local version vector table for the assembled TEXT file.

```
vmfrep1 ariuxit text 5648A70S {DB2VM|DB2VMSFS}
  (logmod modid nocopy
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

modid is the new local modification identifier for the part (for example, L0001).

Step 7. Assemble the File

Use the VMFHLASM command to assemble ARIUXIT.

```
vmfhlasm ariuxit 5648A70S {DB2VM|DB2VMSFS}
  (nockgen $select outmode localsam
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

Notes:

1. Other options are available for the assemble commands. Consult the *VM/ESA: VMSES/E Introduction and Reference* for additional information.
2. We recommend that you use the VMFHLASM command supplied by VMSES/E. Also available are the VMFHASM and VMFASM commands.
3. If the assemble function is successful, the ARIUXIT TXTLnnnn file is placed on the LOCALSAM 2C2(E) disk.

Step 8. Build your New Local Modification

Build your new local modification on the test build disks.

```
vmfbld ppf 5648A70S {DB2VM|DB2VMSFS} (status
vmfbld ppf 5648A70S {DB2VM|DB2VMSFS} (serviced
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

After you issue the VMFBLD command with the STATUS option, use the VMFVIEW BUILD command to view the \$VMFBLD \$MSGLOG file to see which objects will be built.

Step 9. Place the New Local Modification into Production

Use ARISINST to place the new local modification into production.

```
arisinst c 5648A70S {DB2VM|DB2VMSFS}
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

The C function of ARISINST copies the test service and production build disks to the SQLMACH production service and production build disks.

Step 10. Restart the Application Server

Restart the application server in multiple user mode with the required PROTOCOL parameter.

Service Considerations for ARIUXIT

The dummy version of ARIUXIT is not serviceable; other portions of the DSC component, however, are serviceable. If service is applied to any portion of DSC, it is link-edited again. If you have coded your own version of ARIUXIT and completed the previous steps, your version of ARIUXIT will be included in the DSC component.

Defining Your Own Datetime Format

The database manager supports many datetime formats. This section describes the datetime formats and how you can add your own by coding your own exit.

Datetime Formats

The database manager supports DATE, TIME, and TIMESTAMP data types and operations. You can enter a date or a time using many different formats.

Dates can be entered in any of the formats shown in Table 26.

Format Name	Abbreviation	Date Format	Example
International Standards Organization	ISO	yyyy-mm-dd	1993-12-31
IBM USA standard	USA	mm/dd/yyyy	12/31/1993
IBM European standard	EUR	dd.mm.yyyy	31.12.1993
Japanese Industrial Standard Christian Era	JIS	yyyy-mm-dd	1993-12-31
Site-defined	LOCAL	Any site-defined form	—

Times can be entered in any of the formats shown in Table 27 on page 369.

<i>Table 27. Time Formats</i>			
Format Name	Abbreviation	Time Format	Example
International Standards Organization	ISO	<i>hh.mm[.ss]</i>	13.30.05
IBM USA standard	USA	<i>hh:mm AM or PM</i>	1:30 PM
IBM European standard	EUR	<i>hh.mm[.ss]</i>	13.30.05
Japanese Industrial Standard Christian Era	JIS	<i>hh:mm[.ss]</i>	13:30:05
Site-defined	LOCAL	Any site-defined form	—

To define the LOCAL format, you have to code your own date or time exit. For information about coding your own datetime exit, see “Coding Your Own Datetime Exit” on page 372.

Default Output Format

When the database manager is installed, the default date and time formats are both ISO. To change them, you must change the entry in the SYSTEM.SYSOPTIONS table. You must have DBA authority to do this.

For example, to specify that the date output format is USA, enter:

```
UPDATE SYSTEM.SYSOPTIONS -
  SET VALUE='USA' WHERE SQLOPTION='DATE'
```

Similarly, to specify that the time output format is JIS, you enter:

```
UPDATE SYSTEM.SYSOPTIONS -
  SET VALUE='JIS' WHERE SQLOPTION='TIME'
```

Alternatively, you can update the SYSTEM.SYSOPTIONS table by modifying the IBM-supplied ARISDTM MACRO to specify your datetime defaults, then start the DBS utility, specifying the ARISDTM MACRO file as the control file. To modify the ARISDTM MACRO, use the VMSES/E full part replacement local modification procedure. For details, see the appendix in the *DB2 Server for VM Program Directory* describing local modifications.

How Datetime Exits Work

Two datetime installation replaceable exits are provided to allow you to convert datetime values in any installation-defined format into ISO format, or from ISO format into any installation-defined format. These exits which are link-edited into the exit router component ARIXSXR, are called ARIUXDT and ARIUXTM for date and time, respectively.

When the database manager is installed, ARIXSXR is loaded and addressability to the exits is set.

The entries in the SYSTEM.SYSOPTIONS catalog table are used by the database manager to determine the default datetime format for output.

If SYSTEM.SYSOPTIONS indicates that local datetime exits are present, the exits are called during SQL statement processing when conversion between internal and external forms is required.

The product-supplied exits return a -1 return code, meaning the exits have not been replaced by the user exits. If a user program issues an SQL statement that calls the exits, SQLCODE -185 is returned. Therefore, if the user is to replace the exits, the -1 return code must **not** be used.

When Date and Time Exits are Called (Exit Points)

If a program has been preprocessed with the LOCAL format, or if the installation default is LOCAL, then the datetime exits are called before any interpretation of the datetime data values. Otherwise, the database manager attempts to interpret the datetime data values first. In this situation, it calls the local exit only if it does not recognize the datetime value.

The datetime exits are called at the following times:

- When you convert the external form to an internal form:
 - When datetime data is entered by INSERT or UPDATE statements, or by the DATALOAD commands of the DBS utility, or by the INPUT command of ISQL.
 - When a constant or host variable is compared with a DATE or TIME column. The constant can be converted during preprocessing time.
 - When the DATE or TIME scalar functions are used with a string representation of a date or time.

The exit should then convert the installation-defined format into ISO format. The ISO format is then validated and converted into an internal format to be entered into the column or used in comparisons. If the column is a key column for an index, the index entry is made in an internal format.

- When you convert the internal form to an external form:
 - When data is retrieved from the column by SELECT or FETCH statements, or by the DATAUNLOAD commands in the DBS utility, and the default format is local.
 - When the CHAR scalar function is used with the LOCAL format specification.

At this point, the exit should convert the value from ISO format into installation-defined format; then the database manager returns the converted value. In this situation, the exit is called after any edit routine or sort.

The exits are called in the AMODE that the database manager is running in. If the exits do not support 31-bit addressing, the database manager must be started with the AMODE(24) parameter. If the database manager is running in AMODE(24), the load module for the exit must be generated with RMODE 24.

When the exits are called, the registers are set as follows:

Register 0

Undefined.

Register 1

Points to a pointer to the parameter list for ARIUXDT (or ARIUXTM). The format of the parameter list is discussed below. The first field in it is an eye-catcher value.

Register 2—12

Undefined.

Register 13

Points to a standard register save area.

Register 14

Contains the return address.

Register 15

Contains the entry point of the user installation routine.

Registers 2—13 must be saved and restored by the exit. If this is not done, the results will be unpredictable.

Table 28 shows what is in the parameter list used by the date and time exits (see Register 1).

Length	Description
2 words 1 word	Eye-catcher: ARIUXDT or ARIUXTM Length of parameter list
1 word 1 word	Pointer to Function Number Pointer to length of Function Number
1 word 1 word	Pointer to Exit Global Area Pointer to length of Exit Global Area
1 word 1 word	Pointer to ISO Datetime Area Pointer to length of ISO Datetime Area
1 word 1 word	Pointer to LOCAL Datetime Area Pointer to length of LOCAL Datetime Area
1 word 1 word	Pointer to User Work Area Pointer to length of User Work Area
1 word 1 word	Pointer to Environment Dependent Area Pointer to length of Environment Dependent Area
1 word 1 word	Pointer to Exit Return Code Area Pointer to length of Exit Return Code Area

Each area in the parameter list is described below.

- The Eye-catcher and Length of list is initialized by the database manager.
- The Function Number is a full word number describing the function to be performed, as follows:

Number	Function
00000004 00000008	DATE Functions: Convert DATE from LOCAL format to ISO format. Convert DATE from ISO format to Installation format.
00000004 00000008	TIME Functions: Convert TIME from LOCAL format to ISO format. Convert TIME from ISO format to Installation format.

- The EXIT Global Area is not used. Both values are set to zero.
- The length of the ISO Date and Time Areas are 10 bytes and 8 bytes, respectively.
- The length of the LOCAL Date and Time Areas are as defined in the SYSTEM.SYSOPTIONS catalog table. The pointer to the length of the area points to a fullword that contains the value in this table.
- The User Work Area is a 512-byte area.
- The Environment Dependent Area is a 40-byte area. For the datetime exits, only byte 2 is used. It contains D for VSE, and V for VM.
- The Exit Return Code Area is a full word to be set by the exit to the return code.

The possible return codes are:

- 1** The exit supplied by the database manager has not been replaced by a user exit. The database manager then sets SQLCODE to -185.
- 0** The function has been performed.
- 4** Invalid date or time value. The database manager then sets SQLCODE to -181.
- 8** Date or time value not in valid format. The database manager then sets SQLCODE to -180.
- Other** Error in exit. The function number of the exit will be stored in SQLERRD5, and the return code in SQLERRD1. The database manager then sets SQLCODE to -816.

The exit name, function code and return code are set up as message tokens in SQLERRM; they are used when the message associated with SQLCODE -816 is displayed, for example, by the DBS utility and ISQL.

If a program has been preprocessed with the LOCAL format, or if the installation default is LOCAL, then the database manager evaluates the output of the datetime exit if the return code is either 0 or 8. Otherwise, the output is evaluated only if the return code is 0.

Coding Your Own Datetime Exit

User-coded exits must conform to the following:

- The installation replaceable exits must be coded in Assembler language.
- The exits must be reentrant; they must save registers at entry and restore them before exit.

- The exits (and any of the modules they call) must not use any DB2 Server for VM facilities.
- The exits must not use the return code -1.
- When formatting ISO datetime to LOCAL datetime, the user is responsible for formatting the full buffer (the number of bytes equal to the length of the local datetime as defined in the SYSTEM.SYSOPTIONS catalog table).

Figure 113 on page 374 shows the IBM-supplied ARIUXDT module, which is on the service disk. Its file name and file type are ARIUXDT MACRO. You need to modify this source code to support your local date format requirements.

```

          TITLE ' ARIUXDT'
*****
* ARIUXDT USER DATE CONVERSION ROUTINE
* REGISTER ASSUMPTIONS:
*   R1  -> PARMLIST
*   R13 -> SAVE AREA
*   R14 -> RETURN ADDRESS
*   R15 -> ENTRY POINT
*
* ALTHOUGH PROVIDED IN A GENERAL INTERFACE LIBRARY, ARIUXIT IS NOT TO
* BE USED AS A GENERAL PROGRAMMING INTERFACE. REFER TO PRODUCT
* DOCUMENTATION TO DETERMINE INTENDED USAGE.
*****
ARIUXDT CSECT ,
ARIUXDT AMODE 31
ARIUXDT RMODE ANY
        USING *,R15          ESTABLISH TEMP ADDRESSABILITY
        B   PROLOG           BRANCH TO START OF PROGRAM
        DC  C'ARIUXDT'
        DROP R15             DROP R15 AND USE OWN ADDRESSABIL-
*                               ITY
PROLOG  STM  R14,R12,12(R13)  SAVE REGS IN CALLER'S AREA
        LR  R12,R15          SAVE BASE REGISTER
PSTART EQU  ARIUXDT          START OF PROGRAM
        USING PSTART,R12     SET UP BASE REGISTER
        L   R1,0(R1)         POINT TO THE PARAMETER LIST
        USING PARMLIST,R1    ADDRESSABILITY FOR INPUT PARMS
        L   R2,FNPTR         POINT TO FUNCTION TYPE
*****
* M A I N L I N E
*****
MAINLINE DS  0H              START OF CODE
        SPACE
        SR  R15,R15          INITIALIZE RETURN CODE TO ZERO
*****
* HERE YOU WOULD PLACE CODE THAT GETS AND VERIFIES YOUR
* INPUT DATE AND CONVERTS IT TO EITHER TO LOCAL FORMAT OR ISO FORMAT
* A RETURN CODE OF -1 MEANS AN EXIT IS NOT PROVIDED
* A RETURN CODE OF 0 MEANS CONVERSION WAS SUCCESSFUL
* A RETURN CODE OF 4 MEANS THAT THE DATE VALUE WAS OUT OF RANGE
* A RETURN CODE OF 8 MEANS THAT THE DATE WAS INVALID
*****
        BCTR R15,R0          EXIT NOT PROVIDED
        B   RETURN          CONVERSION COMPLETE

```

Figure 113 (Part 1 of 2). IBM-Supplied Version of ARIUXDT


```

*****
* RETURN TO CALLER
*****
RETURN DS 0H RETURN POINT
        L R2,RETPTR LOAD RETCODE PTR
        ST R15,0(R2) STORE EXIT RETURN CODE
        L R14,12(,R13) RESTORE R14
        LM R0,R12,20(R13) RESTORE REST OF CALLER'S REGS
        BR R14 RETURN TO CALLER
        EJECT
PARMLIST DSECT , INPUT PARAMETER LIST
EYECATCH DS CL8 EYECATCHER
PLEN DS F LENGTH OF PARAMETER LIST
FNPTR DS AL4 POINTER TO FUNCTION TYPE
FNLENP DS AL4 LENGTH OF FUNCTION TYPE
GLBPTR DS AL4 POINTER TO GLOBAL EXIT AREA
GLBLENP DS AL4 LENGTH OF GLOBAL EXIT AREA
ISOPTR DS AL4 POINTER TO ISO DATETIME AREA
ISOLENP DS AL4 LENGTH OF ISO DATETIME AREA
LOCPTR DS AL4 POINTER TO LOCAL DATETIME AREA
LOCLENP DS AL4 LENGTH OF LOCAL DATETIME AREA
WORKPTR DS AL4 POINTER TO USER WORK AREA
WORKLENP DS AL4 LENGTH OF USER WORK AREA
ENVPTR DS AL4 POINTER TO ENVIR. DEPENDANT AREA
ENVLENP DS AL4 LENGTH OF ENVIR. DEPENDANT AREA
RETPTR DS AL4 POINTER TO RETURN CODE AREA
RETLENP DS AL4 LENGTH OF RETURN CODE AREA
        EJECT
ARIUXDT CSECT ,
R0 EQU 00 EQUATES FOR REGISTERS 0-15
R1 EQU 01
R2 EQU 02
R3 EQU 03
R4 EQU 04
R5 EQU 05
R6 EQU 06
R7 EQU 07
R8 EQU 08
R9 EQU 09
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
        END ARIUXDT

```

Figure 113 (Part 2 of 2). IBM-Supplied Version of ARIUXDT

Figure 114 on page 376 shows the IBM-supplied ARIUXTM module. This module is on the service disk. Its file name and file type are ARIUXTM MACRO. You can modify this source code to support your local time format requirements.

```

          TITLE ' ARIUXTM'
*****
* ARIUXTM USER TIME CONVERSION ROUTINE
* REGISTER ASSUMPTIONS:
*   R1  -> PARMLIST
*   R13 -> SAVE AREA
*   R14 -> RETURN ADDRESS
*   R15 -> ENTRY POINT
*
* ALTHOUGH PROVIDED IN A GENERAL INTERFACE LIBRARY, ARIUXTM IS NOT TO
* BE USED AS A GENERAL PROGRAMMING INTERFACE. REFER TO PRODUCT
* DOCUMENTATION TO DETERMINE INTENDED USAGE.
*****
ARIUXTM CSECT ,
ARIUXTM AMODE 31
ARIUXTM RMODE ANY
        USING *,R15          ESTABLISH TEMP ADDRESSABILITY
        B   PROLOG          BRANCH TO START OF PROGRAM
        DC   C'ARIUXTM'
        DROP R15           DROP R15 AND USE OWN ADDRESSABILITY
*
PROLOG  STM  R14,R12,12(R13)  SAVE REGS IN CALLER'S AREA
        LR  R12,R15         SAVE BASE REGISTER
PSTART EQU  ARIUXTM         START OF PROGRAM
        USING PSTART,R12    SET UP BASE REGISTER
        L   R1,0(R1)        POINT TO PARAMETER LIST
        USING PARMLIST,R1   ADDRESSABILITY FOR INPUT PARMS
        L   R2,FNPNTR       POINT TO FUNCTION TYPE
*****
* M A I N L I N E
*****
MAINLINE DS  0H           START OF CODE
        SPACE
        SR  R15,R15        INITIALIZE RETURN CODE TO ZERO
*****
* HERE YOU WOULD PLACE CODE THAT GETS AND VERIFIES YOUR
* INPUT TIME AND CONVERTS IT TO EITHER TO LOCAL FORMAT OR ISO FORMAT
* A RETURN CODE OF -1 MEANS AN EXIT IS NOT PROVIDED
* A RETURN CODE OF 0 MEANS CONVERSION WAS SUCCESSFUL
* A RETURN CODE OF 4 MEANS THAT THE TIME VALUE WAS OUT OF RANGE
* A RETURN CODE OF 8 MEANS THAT THE TIME WAS INVALID
*****
        BCTR R15,R0        EXIT NOT PROVIDED
        B   RETURN        CONVERSION COMPLETE

```

Figure 114 (Part 1 of 2). IBM-Supplied Version of ARIUXTM

```

*****
* RETURN TO CALLER
*****
RETURN DS 0H RETURN POINT
        L R2,RETPTR LOAD RETCODE PTR
        ST R15,0(R2) STORE EXIT RETURN CODE
        L R14,12(,R13) RESTORE R14
        LM R0,R12,20(R13) RESTORE REST OF CALLER'S REGS
        BR R14 RETURN TO CALLER
        EJECT
PARMLIST DSECT , INPUT PARAMETER LIST
EYECATCH DS CL8 EYECATCHER
PLEN DS F LENGTH OF PARAMETER LIST
FNPTR DS AL4 POINTER TO FUNCTION TYPE
FNLENP DS AL4 LENGTH OF FUNCTION TYPE
GLBPTR DS AL4 POINTER TO GLOBAL EXIT AREA
GLBLENP DS AL4 LENGTH OF GLOBAL EXIT AREA
ISOPTR DS AL4 POINTER TO ISO DATETIME AREA
ISOLENP DS AL4 LENGTH OF ISO DATETIME AREA
LOCPTR DS AL4 POINTER TO LOCAL DATETIME AREA
LOCLENP DS AL4 LENGTH OF LOCAL DATETIME AREA
WORKPTR DS AL4 POINTER TO USER WORK AREA
WORLENP DS AL4 LENGTH OF USER WORK AREA
ENVPTR DS AL4 POINTER TO ENVIR. DEPENDANT AREA
ENVLENP DS AL4 LENGTH OF ENVIR. DEPENDANT AREA
RETPTR DS AL4 POINTER TO RETURN CODE AREA
RETLENP DS AL4 LENGTH OF RETURN CODE AREA
        EJECT
ARIUXTM CSECT ,
R0 EQU 00 EQUATES FOR REGISTERS 0-15
R1 EQU 01
R2 EQU 02
R3 EQU 03
R4 EQU 04
R5 EQU 05
R6 EQU 06
R7 EQU 07
R8 EQU 08
R9 EQU 09
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
        END ARIUXTM

```

Figure 114 (Part 2 of 2). IBM-Supplied Version of ARIUXTM

After the program is coded, assemble it as you would any other program.

Installing Your Version of ARIUXDT or ARIUXTM

During customization of DB2 Server for VM, you may have to modify TEXT files that are serviced as full part replacement (for example, for user exits, such as ARIUXDT and ARIUXTM). The source is shipped as a MACRO file.

Use the following steps to create and build the affected objects with the new parts. These steps use the VMSES/E local modification procedure.

Step 1. Stop the Application Server

Stop the application server using your normal operating procedures.

Step 2. Log on to the Installation or Service User ID

Log on to the installation or service user ID, 5648A70S.

Step 3. Establish the Minidisk or SFS Directory Order

Establish the access order.

```
vmfsetup 5648A70S {DB2VM|DB2VMSFS}
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

If you have your own PPF override, then substitute that name for the 5648A70S name shown in this command and any following commands.

Step 4. Copy the MACRO Source Code (First Time Only)

Copy the MACRO source code to the local modification disk as ASSEMBLE. This only needs to be done the first time you apply local modifications to this part; otherwise, you will delete your version of the ASSEMBLE file.

```
vmfrep1 {ariuxdt|ariuxtm} macro 5648A70S {DB2VM|DB2VMSFS}  
(filetype assemble outmode localsam
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

Step 5. Edit the ASSEMBLE File

Edit the ARIUXDT ASSEMBLE or the ARIUXTM ASSEMBLE file on the local modification disk (2C2) and make your changes.

Step 6. Update the Local Version Vector Table

Update the local version vector table for the assembled TEXT file.

```
vmfrep1 {ariuxdt|ariuxtm} text 5648A70S {DB2VM|DB2VMSFS}  
(logmod modid nocopy
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

modid is the new local modification identifier for the part (for example, L0001).

Step 7. Assemble the File

Use the VMFHLASM command to assemble ARIUXDT or ARIUXTM.

```
vmfh1asm {ariuxdt|ariuxtm} 5648A70S {DB2VM|DB2VMSFS}  
  (nockgen $select outmode localsam
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

Notes:

1. Other options are available for the assemble commands. Consult the *VM/ESA: VMSES/E Introduction and Reference* for additional information.
2. We recommend that you use the VMFHLASM command supplied by VMSES/E. Also available are the VMFHASM and VMFASM commands.
3. If the assemble function is successful, the ARIUXDT TXTLnnnn file or the ARIUXTM TXTLnnnn file is placed on the LOCALSAM 2C2(E) disk.

Step 8. Build your New Local Modification

Build your new local modification on the test build disks.

```
vmfbld ppf 5648A70S {DB2VM|DB2VMSFS} (status  
vmfbld ppf 5648A70S {DB2VM|DB2VMSFS} (serviced
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

After you issue the VMFBLD command with the STATUS option, you can use the VMFVIEW BUILD command to view the \$VMFBLD \$MSGLOG file to see which objects will be built.

Step 9. Place the New Local Modification into Production

Use ARISINST to place the new local modification into production.

```
arisinst c 5648A70S {DB2VM|DB2VMSFS}
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

The C function of ARISINST copies the test service and production build disks to the database machine (SQLMACH) production service and production build disks.

Step 10. Restart the Application Server

Restart the application server in multiple user mode with the required PROTOCOL parameter.

Updating the SYSTEM.SYSOPTIONS Catalog Table

You need to update the SYSTEM.SYSOPTIONS catalog table to specify the length of your local datetime format.

If you installed a local date or time format, you can update the local date or time length by using the database manager. For example, if the length of your local date format is 10 bytes, enter:

```
UPDATE SYSTEM.SYSOPTIONS -  
  SET VALUE = '10' -  
  WHERE SQLOPTION = 'LDATELEN'
```

The local date length specified must be greater than 9 and less than 255.

If the length of your local time format is 8 bytes, enter:

```
UPDATE SYSTEM.SYSOPTIONS -  
  SET VALUE = '8' -  
  WHERE SQLOPTION = 'LTIMELEN'
```

The local time length specified must be greater than 7 and less than 255.

The changes will be in effect the next time the application server is started.

You can also update the SYSTEM.SYSOPTIONS table by modifying the IBM-supplied ARISDTM MACRO to specify your datetime defaults, then call the DBS utility, specifying the ARISDTM MACRO file as the control file. To modify the ARISDTM MACRO, use the VMSES/E full part replacement local modification procedure. For details on local modifications, see the appendix in the *DB2 Server for VM Program Directory*.

Coding Your Own TRANSPROC Exit

General-Use Programming Interface

The TRANSPROC exit is a General-Use programming interface. General-Use programming interface is defined in “Programming Interface Information” on page iii.

The TRANSPROC exit is used for DBCS conversion. The database manager converts DBCS characters from one DBCS CCSID to another by using the value specified in the TRANSPROC column of the SYSTEM.SYSSTRINGS catalog table. This conversion can be performed when the CCSID of the source and the target are both mixed or are both graphic; that is, the TRANSTYPE column of SYSTEM.SYSSTRINGS has a value of 'PM', 'MM', or 'GG'.

The TRANSPROC exit is also used

- for EUC conversion, to convert MBCS data to mixed data. In EUC conversions, the TRANSTYPE column is either 'PM', or 'GG'.
- to convert Unicode data to CCSIDs that are supported on the database manager. For Unicode to host conversions, the TRANSTYPE column is one of 'US', 'UM', 'UG', or 'UI'.

If you have created your own DBCS CCSIDs, you must create your own conversion routine. To do so:

1. Compile, link-edit and GENMOD your routine to create a MODULE file, and store the module on the production disk.

2. Insert the name of the phase in the TRANSPROC column of the row for which you want either mixed-to-mixed or graphic-to-graphic conversion. (For example, you could create and run a DBSU job to perform this task.)
3. Stop the application server.
4. Run ARISPROD WRITE to be sure you have write access to the production disk.
5. Run the ARISDBMA EXEC with component CCSID. Copy the three CMS files over to the production minidisk, changing their file type from "FLATFILE" to "MACRO." See the *DB2 Server for VM Program Directory* for information on the ARISDBMA EXEC.
6. Restart the application server.
7. All users must reaccess the production minidisk.

The interface between the database manager and a DBCS conversion routine supplied by a user must conform to the following:

- Register conventions:
 - Register 0 is undefined.
 - Register 1 contains the address of the control block that contains the parameters.
 - Registers 2—12 are undefined.
 - Register 13 contains the address of a standard register save area.
 - Register 14 contains the return address.
 - Register 15 contains the address of the user routine.

Registers 2 to 13 must be saved and restored by the routine. If this is not done, the results are unpredictable.

- Parameter list, which is in the following form:
 - Address of the data to be converted (4 bytes)
 - Address of the target for the converted data (4 bytes)
 - Size of the source data (2 bytes)
 - Size of the target area (2 bytes)
 - Return code of the routine (4 bytes).

The TRANSPROC is called in the AMODE that the database manager is running in. If the TRANSPROC does not support 31-bit addressing, the database manager must be started with the AMODE(24) parameter. If the database manager is running in AMODE(24), the load module for the TRANSPROC must be generated with RMODE 24.

The database manager ensures that the size of the target area is at least as large as that of the source data, and that the size of the source data is always an even number. The routine supplied by the user should only convert the source data and put it in the target area. The database manager should do all other operations, such as padding the target area after data conversion is complete. You should also ensure that the routine supplies a nonzero return code if the conversion fails. The routine that you code should not have the same name as any of the defaults

supplied by the database manager for the TRANSPROC column. Figure 115 on page 382 shows the shell for a TRANSPROC routine.

```

      TITLE 'DBCSCONV'
*****
* DBCSCONV USER DBCS CONVERSION ROUTINE
*   REGISTER ASSUMPTIONS:
*     R1  -> PARMLIST
*     R13 -> SAVE AREA
*     R14 -> RETURN ADDRESS
*     R15 -> ENTRY POINT
*
* THIS ROUTINE SHOWS THE INTERFACE TO DB2 Server for VM
*****
DBCSCONV CSECT ,
| DBCSCONV AMODE 31
| DBCSCONV RMODE ANY
      USING *,R15          ESTABLISH TEMP ADDRESSABILITY
      B   PROLOG          BRANCH TO START OF PROGRAM
      DC  C'DBCSCONV'
      DROP R15           DROP R15 AND USE OWN ADDRESSABIL
                          ITY
PROLOG  STM  R14,R12,12(R13)  SAVE REGS IN CALLER'S AREA
        LR  R12,R15         SAVE BASE REGISTER
PSTART EQU  DBCSCONV        START OF PROGRAM
        USING PSTART,R12    SET UP BASE REGISTER
        L   R1,0(R1)        POINT TO PARAMETER LIST
        USING PARMLIST,R1   ADDRESSABILITY FOR INPUT PARMS

```

Figure 115 (Part 1 of 2). TRANSPROC Shell


```

*****
* MAINLINE
*****
MAINLINE DS 0H START OF CODE
        SPACE
*****
* HERE YOU PLACE THE CODE THAT CONVERTS THE INPUT DBCS STRING AND
* PLACES THE CONVERTED STRING IN THE TARGET AREA.
* A NONZERO RETURN CODE INDICATES AN ERROR.
*****
RETURN DS 0H RETURN POINT
        L R14,12(,R13) RESTORE R14
        LM R0,R12,20(R13) RESTORE REST OF CALLER'S REGS
        BR R14 RETURN TO CALLER
        EJECT
PARMLIST DSECT , INPUT PARAMETER LIST
INPTR DS F POINTER TO INPUT STRING
OUTPTR DS F POINTER TO TARGET AREA
INLEN DS H LENGTH OF INPUT STRING
OUTLEN DS H SIZE OF TARGET AREA
RC DS F RETURN CODE
        EJECT
DBCSCONV CSECT ,
R0 EQU 00 EQUATES FOR REGISTERS 0-15
R1 EQU 01
R2 EQU 02
R3 EQU 03
R4 EQU 04
R5 EQU 05
R6 EQU 06
R7 EQU 07
R8 EQU 08
R9 EQU 09
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
END DBCSCONV

```

Figure 115 (Part 2 of 2). TRANSPROC Shell

End of General-Use Programming Interface

Coding Your Own Cancel Exit

In multiple user mode, the resource adapter provides a cancel function that gets control when the terminal operator issues the SQLHX command. The SQLHX command causes an immediate IUCV or APPC/VM sever to be done. The cancel function causes the database manager to do a ROLLBACK WORK on the current in-progress command or logical unit of work. If an explicit SQL CONNECT was done before the SQLHX command was issued, the IUCV or APPC/VM SEVER done on behalf of the SQLHX command causes the user ID to revert to the virtual

machine user ID and the default application server. This is the user ID that is used for the VM implicit CONNECT support. To reestablish the desired user ID, the explicit SQL CONNECT must be reissued. An installation exit can be established that will be given control before the rollback. This installation exit can, at that time, override the SQLHX request, or choose to let the rollback processing continue. If a user enters SQLHX, is using CMS work units, and has more than one logical unit of work that is uncommitted, only the active logical unit of work is rolled back. The suspended CMS work units maintain their current status and position.

When coding your own interactive program to process SQL statements, you can replace the values supplied by the database manager cancel function with your own.

Resource Adapter Cancel Support

The resource adapter provides interactive applications the ability to discontinue processing of an SQL request. If a user issues an SQL statement and wants to cancel processing on that statement, he or she can accomplish this by typing SQLHX on the terminal. The SQLHX command causes an immediate IUCV or APPC/VM SEVER to be done. This SEVER causes the running logical unit of work to be rolled back.

If an explicit SQL CONNECT was done before the SQLHX command was issued, the IUCV or APPC/VM SEVER done on behalf of the SQLHX command causes the user ID to revert to the virtual machine user ID and the default application server. This is the user ID that is used for the VM implicit CONNECT support. To reestablish the desired user ID, the explicit SQL CONNECT must be reissued.

For more information on application usage and modification of resource adapter support, see "RMXC (Resource Adapter Cancel Exit Control)."

RMXC (Resource Adapter Cancel Exit Control)

This control block controls the resource adapter cancel support. It is used by the immediate command exit supplied with the resource adapter as well as by the resource adapter cancel support. An A within the field description indicates that the field is set by the application. An R indicates that the resource adapter sets the field. The resource adapter allocates storage for the RMXC, while the ARIRCAN macro provides application addressability to the RMXC.

General-Use Programming Interface

Macro ARIRCAN is a General-Use programming interface. General-Use programming interface is defined in "Programming Interface Information" on page iii.

Dec(HEX) RMAR

0(0)	RMXCEYEC - 'RMXC' (eyecatcher)	
8(8)	RMXCLENF - RMXC length	Reserved
16(10)	RMXCAPPL - For application use	RMXCCXIT - Optional pointer to an application cancel exit routine (binary zero if no exit defined)
24(18)	RMXCCONT (1)	Reserved
32(20)	Reserved	RMXCXC - Exit code (2)
40(28)	Reserved - binary zeros	
48(30)	RMXCECBP → Cancel ECP	RMXCPC - Post mask, DB2/VM cancel
56(38)	RMXCPHC - Post mask, stop processing	Reserved - binary zeros
72(48)	/ /	

(1) Application cancel exit stop indicator:

'Y' to continue cancel
'N' to stop cancel

(2) Exit codes are:

RMXCWLNK: 0 - Resource adapter link wait
 RMXCWSQL: 4 - Resource adapter SQL wait
 RMXCWSC: 8 - Resource adapter wait for SQL COMMIT WORK
 RMXCWSRB: 12 - Resource adapter wait for SQL ROLLBACK WORK
 RMXCWNO: -1 - Resource adapter not waiting
 RMXCCAN: -2 - Resource adapter not waiting and cancel has been requested

Figure 116. Resource Adapter Cancel Exit Control (RMXC)

The resource adapter provides a cancel support for applications by using the VM immediate command extension support. During startup in the multiple user mode environment, the resource adapter establishes an immediate command of SQLHX. When this command is entered, if the resource adapter is waiting for a reply from the virtual machine, the SQL statement that is currently being processed is canceled, and the resource adapter returns to the application with a SQLCODE of -914.

If the SQLHX command is entered and the resource adapter is not waiting for a reply from the database manager, the next SQL statement that is issued is canceled with a return code of -914.

Cancel support is intended for interactive applications such as ISQL, where the terminal operator knows what is being canceled. Applications that perform preplanned SQL queries in a batch type environment can be adversely affected by the arbitrary issuing of the SQLHX command. Also, applications can require that cancel support called by a command other than SQLHX.

To provide flexibility in the resource adapter cancel support, a BAL macro is provided to allow applications to modify the basic cancel support in the following ways:

- Create additional immediate command names with which to call the cancel support command exit.
- Remove the SQLHX immediate command exit, which effectively discontinues the resource adapter cancel support unless the application provides its own cancel exit.

The format of the ARIRCAN macro is:

```

      { ARIRCAN CMDNAME=pointer-address }
      { ARIRCAN TYPE=USER                }

*****
* SAMPLE CANCEL EXIT
*****
      .
      .
      .
CANSAMP DS    0H
        LA    R3,NEWNAME                LOAD ADDR OF CANCEL ALIAS
        ARIRCAN CMDNAME=(R3)
        ARIRCAN TYPE=USER
                                           GIVE NEW NAME TO SQL
      .
      .
      .
NEWNAME DC    CL8'CANCEL'                CANCEL ALIAS NAME

```

The CMDNAME=*pointer-address* is the address to an 8-byte field containing an alternate command name to be used to call the resource adapter immediate command exit. The command name SQLHX remains in effect while the additional command functions as an alias to call the immediate command exit.

The TYPE=USER specifies that the user handles any cancel exit function. It causes the resource adapter to remove the immediate command exit established at startup.

Output from this macro with the CMDNAME or TYPE operands is an RDIIN, type 166. For the CMDNAME operand, the command name pointer is placed in the RDIVPARAM field. A call is also made to ARIPRDI. When the call is made, the resource adapter either establishes an additional immediate command name or removes the SQLHX immediate exit, depending on the parameters specified in the ARIRCAN macro.

On successful return to the application, register 15 points to the RMXC control block. The following are required for the ARIRCAN macro:

- The call must be from a module that has completed the assembler preprocessing. This provides for SQLDSECT and SQLCA addressability.
- As for EXEC SQL, register 13 must point to a standard 72-byte save area, and registers 1, 14, and 15 are modified as a result of the call generated by the macro.

|_____ End of General-Use Programming Interface _____|

The CANCEL function is supported by two parts of the resource adapter:

- The immediate command process (modules ARIRCL1C and ARIRCL2C).
- The mainline resource adapter process that gets notified (posted) by the immediate command process when a cancel is being requested.

The resource adapter invokes user exits in AMODE(31).

When the resource adapter calls ARICCOM to send a message across the path to the database manager, a CMS WAITECB macro is issued for two event control blocks (ECBs). One ECB is posted when the communication is complete. The other is posted by the immediate command process if it is called by the SQLHX command (or any other user-defined command). If the cancel ECB is posted, the resource adapter mainline (ARIRVRM) ends processing of the SQL statement by calling ARICCOM to do a CMS IUCV SEVER. This SEVER causes the database manager to rollback the processing that was taking place for the SQL statement. The resource adapter then returns to the application with a -914 SQLCODE.

The resource adapter provides for a user exit to be called from the mainline cancel process. If the RMXCCXIT field contains a nonzero value, it is assumed that a user exit routine exists. The resource adapter branches to this exit before issuing the IUCV SEVER to cause the cancel. At this point the user exit can discontinue the cancel request by setting the RMXCCONT field to N.

Register contents on entry to the user exit are:

Register 1 Pointer to RMXC control block

Register 13 Pointer to 72-byte save area

Register 14 Return address

Register 15 Entry point of user exit.

It is assumed that registers are restored upon return from the exit.

Field Procedures

General-Use Programming Interface

A field procedure is a General-Use programming interface. Macro ARIBFPPB is a General-Use programming interface. General-Use programming interface is defined in "Programming Interface Information" on page iii.

A field procedure is a user-written exit routine that transforms values in a single short-string column. When values in the column are changed, or new values are inserted, the field procedure is run to encode each value, which is then stored. When values are retrieved from the column, the field procedure is run to decode each value back to the original string value. A field procedure can be used to alter the sorting sequence of values entered in a column. For example, telephone directories sometimes require that names such as McCabe and MacCabe appear next to each other. This cannot be achieved with the standard EBCDIC sorting sequence. Languages that do not use the Roman alphabet have similar requirements. However, if a column is provided with a suitable field procedure, you can obtain the desired ordering with the ORDER BY clause.

Any indexes defined on a column that uses a field procedure are built with encoded values.

The transformation that a field procedure performs on a value is called *field-encoding*. The same routine is used to undo the transformation when values are retrieved; that operation is called *field-decoding*.

The field procedure is called when a table is created or altered, to define the data type and attributes of an encoded value to the database manager. That operation is called *field-definition*. The data type of the encoded value can be CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC. If the datatype is VARCHAR the length must be 254 or less. If the database is VARGRAPHIC, the length must be 127 or less. For the applicable data types, see the description for the FPVDTYPE field in Table 30 on page 395. The length, precision, or scale of the encoded value must be compatible with its data type. Values in columns with a field procedure are described to the database manager in the following catalog tables:

- SYSTEM.SYSCOLUMNS
- SYSTEM.SYSFIELDS
- SYSTEM.SYSFPARMS
- SYSTEM.SYSKEYCOLS.

For more information about catalog tables, see the *DB2 Server for VSE & VM SQL Reference* manual.

Specifying the Field Procedure

To name a field procedure for a column, use the FIELDPROC clause of the CREATE TABLE or ALTER TABLE statement, followed by the name of the procedure and, optionally, a list of parameters. You can use a field procedure only with a short string column. You cannot add a field procedure to an existing column of a table. You can, however, use the ALTER TABLE statement to add to an existing table a new column that uses a field procedure. (To do so, you would have to unload the data, recreate the table, and load the data back into the table.)

The optional parameter list that follows the procedure name is a list of constants, enclosed in parentheses, called the *literal list*. The literal list is incorporated into a data structure called the *field procedure parameter value list* (FPPVL). That structure is passed to the field procedure during the field-definition operation. At that time, the procedure can modify it or return it unchanged. The output form of the FPPVL is called the *modified FPPVL*; it is stored in the DB2 Server for VM catalog as part of the field description. The modified FPPVL is passed again to the field procedure when the procedure is called for field-encoding or field-decoding.

When Field Procedures are Called

A field procedure specified for a column is called in three situations:

- For field-definition, when the CREATE TABLE or ALTER TABLE statement that names the procedure is run. When called, the procedure is expected to:
 - Determine whether the data type and attributes of the column are valid
 - Verify the literal list, and change it if required
 - Provide the field description of the column

- Define the amount of working storage needed by the field-encoding and field-decoding processes.
- For field-encoding, when a column value is to be field-encoded. That occurs for any value that is:
 - Inserted in the column by an SQL INSERT or PUT statement, or loaded by the DBS utility DATALOAD or RELOAD commands
 - Changed by an SQL UPDATE statement
 - Compared to a column with a field procedure, unless the comparison operator is LIKE. The value being encoded is a host variable or constant.
- For field-decoding, when a stored value is to be field-decoded back into its original string value. This occurs for any value that is:
 - Retrieved by an SQL SELECT or FETCH statement, or by the DBS utility DATAUNLOAD or UNLOAD commands
 - Compared to another value with the LIKE comparison operator. The value being decoded is from the column that uses the field procedure.

In this situation the field procedure is called after any DB2 Server for VM sort.

A field procedure is never called to process a null value.

General Considerations for Writing Field Procedures

Your field procedure must adhere to the following rules:

- It must be written in Assembler.
- Its name must not start with ARI, to avoid conflict with the DB2 Server for VM modules.
- It must not call any SVC services or VM services.
- It must store registers in an area pointed to by R13, and restore them before returning.
- It must be serially reusable.
- It must not contain any SQL statements.
- It must reside on a database minidisk and be accessible when the database manager is running.
- The field procedure is called in the AMODE that the database manager is running in. If it does not support 31-bit addressing, the database manager must be started with the AMODE(24) parameter. If the database manager is running in AMODE(24), the load module for the field procedure must be generated with RMODE 24.

Attention: A field procedure should always transform one input data value into one output data value, unless the parameters are different. This means that the same field procedure with the same parameters must implement a one to one data conversion, in both directions. The field-decoding function must be the exact inverse of the field-encoding function. For example, if a routine encodes ALABAMA to 01, it must decode 01 to ALABAMA. A violation of this rule can lead to unpredictable results and possible data corruption.

A Warning about Blanks

When the database manager compares the values of two strings with different lengths, it temporarily pads the shorter string with blanks (in either single-byte or double-byte characters, as appropriate) up to the length of the longer string. If the shorter string is the value of a column with a field procedure, the padding is done to the encoded value, but the pad character is **not** encoded. Hence, if the procedure changes blanks to some other character, encoded blanks at the end of the longer string are not equal to padded blanks at the end of the shorter string. That situation can lead to errors; for example, some strings that should be equal may not be recognized as such. You should **not** encode blanks with a field procedure.

Maintaining Field Procedures

Field procedures should reside as modules with the relocatable information saved (RLD option on the CMS LOAD command) on any server machine minidisk or SFS directory which is accessed when the database manager is running. Field procedures are loaded by the VM NUCXLOAD command. The maximum number of active field procedures on one installation is 16. If this limit is exceeded, an attempt to load a field procedure results in an SQLCODE -682 with reason code 4.

Recovering from Abends in Exits

If a field procedure ends abnormally, a message (ARI0022E) to remove the field procedure from the installation is issued to the operator, the database manager takes a SNAP dump, and processing continues.

Security with Field Procedures

Since exit routines run as extensions of the database manager and have all its privileges, they can impact its security and integrity. All field procedures must be tested and appropriate security measures taken before they are installed on a system.

Field Procedures for Cultural Sorts

By default, string data is sorted based on the S/390 collating sequence. However, the collating sequence required for certain alphabets is different from the default S/390 collating sequence. Users expect that sorted data will match the order that is culturally correct for them and that searches on data will return the result that is correct for the sorting sequence of their language. They are at ease with only one sort order, the one used in their dictionaries, telephone directories, book indices, and so on.

A way to accommodate special sorting requirements is to use Field Procedures. Field Procedures can be used to encode data being inserted into a column. The encoding effectively alters the collating sequence for the data in the column, enabling the special sorting requirements to be met by the S/390 collating sequence.

Two field procedures are provided. The procedures are found on the database machine's service disk.

The field procedures provided are:

- FP870L2 for Slovenia, Poland and Romania
- FP102CY for Russia, Bulgaria, Serbia and Montenegro

The field procedures are written in Assembler. The field procedure must be assembled and the corresponding module must be generated and placed on a disk that is accessible to the database manager when it is running. Note that in VM, the MACLIB FLDPROC (which is provided with DB2 Server for VM) must be specified on the GLOBAL MACLIB statement in order to assemble the field procedure and generate the module.

Once the module for the field procedure has been generated and made accessible to the database manager, it can be used by specifying its name in the FIELDPROC clause of the CREATE TABLE or ALTER TABLE statement.

Field Procedure Considerations with Data Capture for VM

If Data Propagator Capture for VM is being used on tables which have columns with field procedures, “1-way” field procedures must be defined on the Data Propagator Change Data (CD) tables to properly propagate this data. The following example using the FP102CY sample program illustrates what is required.

The sample field procedure FP102CY uses the following encoding table to modify data when it is inserted into a column:

```

*                               EnCode Table for Cyrillic
*                               0 1 2 3 4 5 6 7 8 9 A B C D E F
CPLAE  DC  X'000102030405060708090A0B0C0D0E0F' 0
        DC  X'101112131415161718191A1B1C1D1E1F' 1
        DC  X'202122232425262728292A2B2C2D2E2F' 2
        DC  X'303132333435363738393A3B3C3D3E3F' 3
        DC  X'4064AFADB5B3BBBFC1C5524A60505F47' 4
        DC  X'5BCBD1DFD0E3EDF45CB053585951454C' 5
        DC  X'4349AEB6B4BCC0C2C6CC63445E416248' 6
        DC  X'D2E0DE42E4EEFBA3A54B465D574E614F' 7
        DC  X'E96F71737577797B7D7FABB1E5A9E7BD' 8
        DC  X'C381838587898B8D8F91C7C9CDCFD3D5' 9
        DC  X'FD4D939597999B9D9FA1D7D9DBE1B7A7' A
        DC  X'F7F5B9EFF9F1EBF3FCA4A6EAACB2E6AA' B
        DC  X'5470727476787A7C7E80E8BEC4C8CACE' C
        DC  X'55828486888A8C8E9092D0D4D6FED8DA' D
        DC  X'5A569496989A9C9EA0A2DCE2B8A8F8F6' E
        DC  X'65666768696A6B6C6D6EBAF0FAF2ECFF' F
*                               0 1 2 3 4 5 6 7 8 9 A B C D E F
*

```

Assume Table T1 is created with column C1 with field procedure FP102CY defined on it. When the value '50'X is inserted into column C1, the database will invoke the encoding procedure of FP102CY which will encode the value '50'X to '5B'X. The log record written to the database log for this Insert operation will indicate that the value '5B'X was inserted into column C1. When Capture processes this log record, it will insert a record into the Change Data (CD) table indicating that the value '5B'X was inserted. When DataPropagator Apply is ready to propagate data for this table, Apply will insert the value '5B'X into the target table. The data in the target table does not match the data in the source table.

The recommended method to correct this situation is to define “1-way” field procedures based on the original field procedure. The only change required is to modify the encoding procedure of FP102CY so that the data is not modified when it is inserted. The decoding procedure will decode data in the same way as the

original field procedure. For example, FP102CY1, will be exactly the same except, the encoding table will be changed to:

```

*           EnCode Table for Cyrillic
*           0 1 2 3 4 5 6 7 8 9 A B C D E F
CPLAE     DC   X'000102030405060708090A0B0C0D0E0F' 0
           DC   X'101112131415161718191A1B1C1D1E1F' 1
           DC   X'202122232425262728292A2B2C2D2E2F' 2
           DC   X'303132333435363738393A3B3C3D3E3F' 3
           DC   X'404142434445464748494A4B4C4D4E4F' 4
           DC   X'505152535455555758595A5B5C5D5E5F' 5
           DC   X'606162636465666768696A6B6C6D6E6F' 6
           DC   X'707172737475767778797A7B7C7D7E7F' 7
           DC   X'808182838485868788898A8B8C8D8E8F' 8
           DC   X'909192939495969798999A9B9C9D9E9F' 9
           DC   X'A0A1A2A3A4A5A6A7A8A9AAABACADAFAF' A
           DC   X'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF' B
           DC   X'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF' C
           DC   X'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF' D
           DC   X'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF' E
           DC   X'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF' F
*           0 1 2 3 4 5 6 7 8 9 A B C D E F
*

```

When T1 is defined to as a subscription source table for DataPropagator, the generated SQL statements must be modified before being used to define the CD table. The generated SQL statements should be modified to add the FP102CY1 field procedure to any columns based on columns that have FP102CY defined on them. The modified SQL statements should then be run to create the CD table.

Now that FP102CY1 is being used on the CD table the value '50'X will be propagated correctly. As before, a log record with the value '5B'X will be processed by Capture. When Capture inserts this value into the CD table, FP102CY1 will be called to encode this value. The encoding procedure of FP102CY1 will be called, and '5B'X will be encoded to the value '5B'X. When Apply reads this value to propagate it, the decoding procedure of FP102CY1 will decode '5B'X to the value '50'X, and the value '50'X will be inserted into the target table.

If the target table requires the cultural sort on this column, the target table columns should also be defined with field procedure FP102CY.

Field Procedure Interface to the Database Manager

This section describes certain control blocks that are used to communicate to a field procedure, under the following headings:

- “The Field Procedure Parameter List (FPPL)” on page 393
- “The Work Area” on page 393
- “The Field Procedure Information Block (FPIB)” on page 394
- “Value Descriptors” on page 394
- “The Field Procedure Parameter Value List (FPPVL)” on page 395.

The Field Procedure Parameter List (FPPL)

The FPPL is pointed to by register 1 on entry to a field procedure. It, in turn, contains the addresses of five other areas, shown in Figure 117. The FPPL and the areas to which it points are all described by the mapping macro ARIBFPPB, which is provided in FLDPROC MACLIB.

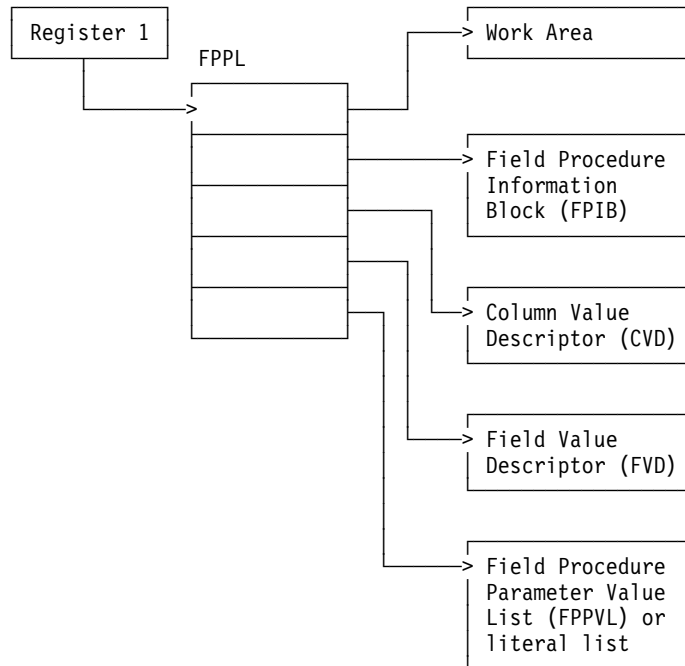


Figure 117. Field Procedure Parameter List

The Work Area

The work area is an area of storage used by a field procedure as working storage. A new area is provided each time the procedure is called.

The size of the area you need depends on the way you have programmed your field-encoding and field-decoding operations. For the field-definition operation, the database manager passes your routine a value of 512 bytes for the length of the work area (FPBWKLN in FPIB). If, for example, the longest work area you need for field-encoding or field-decoding is 1024 bytes, your field-definition operation must change the length to 1024. Thereafter, whenever your field procedure is called for either encoding or decoding, the database manager makes an area of 1024 bytes available to it.

If 512 bytes is sufficient for your operations, your field-definition operation need not change the value supplied by the database manager. If you need less than 512 bytes, your field-definition can return a smaller value. However, your field-definition itself must not use more than 512 bytes.

The Field Procedure Information Block (FPIB)

The FPIB communicates general information to a field procedure. For example, it tells what operation is to be done, allows the field procedure to signal errors, and gives the size of the work area. Its format is shown in Table 29.

Name	'Hex' Offset	Data Type	Description
FPBFCODE	0	Signed halfword integer	Function code. Code Means 0 Field-encoding 4 Field-decoding 8 Field-definition
FPBWKLN	2	Signed halfword integer	Length of work area; the maximum is 32767 bytes.
	4	Signed halfword integer	Reserved.
FPBRTNC	6	Character, 2 bytes	Return code set by field procedure.
FPBR SNC	8	Character, 4 bytes	Reason code set by field procedure.
FPBTOKP	12	Address	Address of a 40-byte area, within the work area or within the field procedure's static area, containing an error message.

Value Descriptors

Value descriptors describe the data type and other attributes of a value. They are used with field procedures in these ways:

- During field definition, they describe each constant in the field procedure parameter value list (FPPVL). The set of these value descriptors is part of the FPPVL control block.
- During field encoding and field decoding, the decoded (column) value and the encoded (field) value are described by the column value descriptor (CVD) and the field value descriptor (FVD).

The CVD contains a description of a column value and, if appropriate, the value itself. During field encoding, the CVD describes the value to be encoded; during field decoding, it describes the decoded value to be supplied by the field procedure; and during field definition, it describes the column as defined in the CREATE TABLE or ALTER TABLE statement.

The FVD contains a description of a field value and, if appropriate, the value itself. During field-encoding, the FVD describes the encoded value to be supplied by the field procedure; during field-decoding, it describes the value to be decoded. Field-definition must put into the FVD a description of the encoded value.

The format of value descriptors is shown in Table 30 on page 395.

Name	'Hex' Offset	Data Type	Description
FPVDTYPE	0	Signed halfword integer	Data type of the value: Code Means 16 CHAR 20 VARCHAR 24 GRAPHIC 28 VARGRAPHIC
FPVDVLEN	2	Signed halfword integer	For a varying-length string value, its maximum length.
FPVDVALE	4	None	The value. If the value is a varying-length string, the first half word is the value's actual length in bytes. This field is not present in a CVD, or in an FVD used as input to the field-definition operation.

The Field Procedure Parameter Value List (FPPVL)

The FPPVL communicates the literal list, supplied in the CREATE TABLE or ALTER TABLE statement, to the field procedure during field definition. At that time the field procedure can reformat the FPPVL. The reformatted FPPVL is stored in SYSTEM.SYSPARMS and communicated to the field procedure during field encoding and field decoding as the modified FPPVL.

Its format is shown in Table 31.

Name	'Hex' Offset	Data Type	Description
FPPVLEN	0	Signed halfword integer	Length in bytes of the area containing FPPVCNT and FPPVVDS. At least 254 for field-definition.
FPPVCNT	2	Signed halfword integer	Number of value descriptors that follow, equal to the number of parameters in the FIELDPROC clause. Zero if no parameters were listed.
FPPVVDS	4	Structure	For each parameter in the FIELDPROC clause, there is: <ul style="list-style-type: none"> • A signed fullword integer giving the length of the following value descriptor. • A value descriptor.

Field-Definition (Function Code 8)

The input provided to the field-definition operation, and the output required, are as follows:

On ENTRY

The registers have the following information:

Register	Contains
1	Address of the field procedure parameter list (FPPL). For a schematic diagram, see Figure 117 on page 393.
2-12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The contents of all other registers, and of fields not listed below, are unpredictable.

The work area consists of 512 contiguous uninitialized bytes.

The FPIB has the following information:

Field	Contains
FPBFCODE	8, the function code.
FPBWKLN	512, the length of the work area.

The CVD has the following information:

Field	Contains										
FPVDTYPE	One of these codes for the data type of the column value: <table><thead><tr><th>Code</th><th>Means</th></tr></thead><tbody><tr><td>16</td><td>CHAR</td></tr><tr><td>20</td><td>VARCHAR</td></tr><tr><td>24</td><td>GRAPHIC</td></tr><tr><td>28</td><td>VARGRAPHIC</td></tr></tbody></table>	Code	Means	16	CHAR	20	VARCHAR	24	GRAPHIC	28	VARGRAPHIC
Code	Means										
16	CHAR										
20	VARCHAR										
24	GRAPHIC										
28	VARGRAPHIC										
FPVDVLEN	The length attribute of the column.										

The FPVDVALE field is omitted.

The FVD provided is 4 bytes long.

The FPPVL has the following information:

Field	Contains
FPPVLEN	The length, in bytes, of the area containing the parameter value list. The minimum value is 254, even if there are no parameters.
FPPVCNT	The number of value descriptors that follow; zero if there are no parameters.
FPPVSDS	A contiguous set of value descriptors, one for each parameter in the parameter value list, each preceded by a 4-byte length field.

On EXIT

The registers must have the following information:

Register	Contains
2-12	The values they contained on entry.
15	The integer zero if the column described in the CVD is valid for the field procedure; otherwise the value must not be zero.

Fields listed below must be set as shown; all other fields must remain as on entry.

The FPIB must have the following information:

Field	Contains
FPBWKLN	The length, in bytes, of the work area to be provided to the field-encoding and field-decoding operations; 0 if no work area is required.
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in an SQL return code of -681, which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

The FVD must have the following information:

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value. Any of the data types listed in Table 30 on page 395 is valid.
FPVDVLEN	The length of the field value.

Field FPVDVALE must **not** be set; the length of the FVD is 4 bytes only.

The FPPVL can be redefined to suit the field procedure, and returned as the modified FPPVL, subject to the following restrictions:

- The field procedure must not increase the length of the FPPVL.
- The FPPVLEN must contain the actual length of the modified FPPVL, or 0 if no parameter list is returned.

The modified FPPVL is recorded in the SYSTEM.SYSFPARMS catalog table and is passed again to the field procedure during field-encoding and field-decoding. The modified FPPVL need not have the format of a field procedure parameter list, and it need not describe constants by value descriptors.

Field-Encoding (Function Code 0)

The input provided to the field-encoding operation, and the output required, are as follows:

On ENTRY

The registers have the following information:

Register	Contains
1	Address of the field procedure parameter list (FPPL). For a schematic diagram, see Figure 117 on page 393.
2-12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.

15 Address of entry point of exit routine.

The contents of all other registers, and of fields not listed below, are unpredictable.

The work area is contiguous, uninitialized, and of the length specified by the field procedure during field-definition.

The FPIB has the following information:

Field	Contains
FPBFCD	0, the function code.
FPBWKLN	The length of the work area.

The CVD has the following information:

Field	Contains
FPVDTYPE	The numeric code for the data type of the column value, as shown in Table 30 on page 395.
FPVDVLEN	The length of the column value.
FPVDVALE	The column value; if the value is a variable-length string, the first halfword contains its length.

The FVD has the following information:

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value.
FPVDVLEN	The length of the field value.
FPVDVALE	An area of unpredictable content that is as long as the field value.

The modified FPPVL produced by the field procedure during field-definition is provided if it exists.

On EXIT

The registers must have the following information:

Register	Contains
2-12	The values they contained on entry.
15	The integer zero if the encoding is successful; otherwise the value must not be zero.

The FVD must contain the encoded (field) value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The FPIB may have the following information:

Field	Contains
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in an SQL return code of -681, which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKPT, are also placed into

the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

All other fields must remain as on entry.

Field-Decoding (Function Code 4)

The input provided to the field-decoding operation, and the output required, are as follows:

On ENTRY

The registers have the following information:

Register	Contains
1	Address of the field procedure parameter list (FPPL). For a schematic diagram, see Figure 117 on page 393.
2-12	Unknown values that must be restored on exit.
13	Address of the register save area.
14	Return address.
15	Address of entry point of exit routine.

The contents of all other registers, and of fields not listed below, are unpredictable.

The work area is contiguous, uninitialized, and of the length specified by the field procedure during field-definition.

The FPIB has the following information:

Field	Contains
FPBFCODE	4, the function code.
FPBWKLN	The length of the work area.

The CVD has the following information:

Field	Contains
FPVDTYPE	The numeric code for the data type of the column value, as shown in Table 30 on page 395.
FPVDVLEN	The length of the column value.
FPVDVALE	An area of unpredictable content that is as long as the column value.

The FVD has the following information:

Field	Contains
FPVDTYPE	The numeric code for the data type of the field value.
FPVDVLEN	The length of the field value.
FPVDVALE	The field value; if the value is a varying-length string, the first halfword contains its length.

The modified FPPVL, produced by the field procedure during field-definition, is provided if it exists.

On EXIT

The registers must have the following information:

Register	Contains
2-12	The values they contained on entry.
15	The integer zero if the decoding is successful; otherwise the value must not be zero.

The CVD must contain the decoded (column) value in field FPVDVALE. If the value is a varying-length string, the first halfword must contain its length.

The FPIB may have the following information:

Field	Contains
FPBRTNC	An optional 2-byte character return code, defined by the field procedure; blanks if no return code is given.
FPBRSNC	An optional 4-byte character reason code, defined by the field procedure; blanks if no reason code is given.
FPBTOKP	Optionally, the address of a 40-byte error message residing in the work area or in the field procedure's static area; zeros if no message is given.

Errors signalled by a field procedure result in an SQL return code of -681, which is set in the SQL communication area (SQLCA). The contents of FPBRTNC and FPBRSNC, and the error message pointed to by FPBTOKP, are also placed into the tokens, in SQLCA, as field SQLERRMT. The meaning of the error message is determined by the field procedure.

All other fields must remain as on entry.

A Sample Exit

Figure 118 on page 401 shows an example of a field procedure.

```

FLCTFLC TITLE 'DB2 Server for VM FIELD PROCEDURE EXAMPLE'
FLCTFLC START 0
FLCTFLC AMODE 31
FLCTFLC RMODE ANY
*****
*          DB2 Server for VM FIELD PROCEDURE TO CONVERT          *
*          FIXED LENGTH CHARACTER TO FIXED                        *
*          LENGTH CHARACTER USING A LOOKUP TABLE                *
*****
          SPACE 3
          PRINT GEN
          USING FLCTFLC,R3          BASE REGISTER
          USING FPIB,R9            COMMON INFORMATION BLOCK
          USING FPVD,R10           VALUE DESCRIPTOR
          USING FPPL,R11           PARAMETER LIST
          USING WA,R12             WORK AREA
          USING FPPVL,R8           PARAMETER VALUE LIST
          USING TBLHDRD,R7         TABLE HEADER
          SPACE 3
*****
*          SET UP MAIN LINE          RETURN R14          *
*****
          SPACE 3
          SAVE (14,12),,FLCTFLC
          LR   R3,R15              LOAD BASE REGISTER
          LR   R11,R1              PARAMETER LIST POINTER
          L    R12,FPPWORK         WORK AREA ADDRESS
          ST   R13,SAVE13
          L    R9,FPPFPIB         COMMON INFORMATION BLOCK
          MVC  FPBRTNC,=AL2(FPBRC0) RETURN CODE = 0
          LH   R2,FPBFCODE
          L    R15,FDLFC(R2)      SELECT APPROPRIATE ROUTINE
          LA   R14,RET1
          BR   R15
RET1     DS   0H
          PACK WADW,FPBRTNC        SET RETURN CODE R15
          CLI  FPBRTNC+L'FPBRTNC-1,C' '
          BNE  NOTBL
          PACK WADW,FPBRTNC(L'FPBRTNC-1)
NOTBL   DS   0H
          CVB  R15,WADW
          L    R13,SAVE13
          RETURN (14,12),T,RC=(15)
          LTORG
FDLFC   DC   A(ENCODE,DECODE,DEFINE)
          SPACE 3

```

Figure 118 (Part 1 of 9). Field Procedure Example

```

*****
*          ENCODING ROUTINE                      RETURN R14          *
*****
          SPACE 3
ENCODE   DS    0H
          MVC   FUNCT,=C'ENCD'
          LA    R5,B1
          B     CHKINP          CHECK INPUT DESCRIPTION
B1       DS    0H
          LA    R5,B2
          B     CHKOUT          CHECK OUTPUT DESCRIPTION
B2       DS    0H
          SPACE 3
*****
*          LOOKUP ROUTINE FOR ENCODING          *
*****
          SPACE 3
          L     R10,FPPCVD          INPUT VALUE
          L     R6,TABADDR          TOP OF LOOKUP TABLE
          LA    R5,B3
          B     SETLUP          SET UP LOOKUP VARIABLES
B3       DS    0H
          SPACE 3
*****
*          SET UP LOOP VARIABLES          *
*****
          SPACE 3
          SR    R4,R4          CLEAR R4
          IC    R4,ILEN          LENGTH FOR COMPARE
          SH    R4,=H'1'          -1
ITOP     DS    0H
          EX    R4,CLCINST
          BE    IHIT
          A     R6,INCRLEN          INCREMENT TO NEXT ENTRY
          BCT   R13,ITOP
          LA    R13,ER5
          B     ERROR4
IHIT     DS    0H
          L     R10,FPPFVD
          SPACE 3

```

Figure 118 (Part 2 of 9). Field Procedure Example

```

*****
*      SET UP MOVE INSTRUCTION      *
*****
      SPACE 3
      SR   R13,R13          CLEAR R13
      IC   R13,OLEN        OUTPUT LENGTH
      SH   R13,=H'1'      -1
      SR   R5,R5          CLEAR R5
      IC   R5,ILEN        INPUT LENGTH
      AR   R6,R5          POINT TO OUTPUT VALUE IN TABLE
      EX   R13,MVCINST
      BR   R14
      SPACE 3
*****
*      MOVE AND COMPARE INSTRUCTION FOR EXECUTION INSTRUCTION  *
*****
      SPACE 3
      DS   0H
CLCINST CLC  0(1,R6),FPVDVALE
MVCINST MVC  FPVDVALE,0(R6)
      SPACE 3
*****
*      DECODING ROUTINE      *
*****
      SPACE 3
DECODE  DS   0H
      MVC  FUNCT,=C'DECD'
      LA   R5,BB1
      B    CHKINP          CHECK INPUT DESCRIPTION
BB1     DS   0H
      LA   R5,BB2
      B    CHKOUT          CHECK OUTPUT DESCRIPTION
BB2     DS   0H
      SPACE 3
*****
*      LOOKUP ROUTINE FOR DECODING      *
*****
      SPACE 3
      L    R10,FPPFVD      OUTPUT VALUE
      L    R6,TABADDR      TOP OF LOOKUP TABLE
      LA   R5,BB3
      B    SETLUP          SET LOOKUP VARIABLES
BB3     DS   0H
      SPACE 3

```

Figure 118 (Part 3 of 9). Field Procedure Example

```

*****
*       SET UP LOOP VARIABLES                               *
*****
      SPACE 3
      SR   R4,R4                CLEAR R4
      IC   R4,OLEN              LENGTH FOR COMPARE
      SH   R4,=H'1'            -1
      SR   R5,R5                CLEAR R5
      IC   R5,ILEN              INPUT LENGTH
      AR   R6,R5                POINT TO OUTPUT VALUE IN TABLE
OTOP   DS   0H
      EX   R4,CLCINST
      BE   OHIT
      A    R6,INCRLEN           POINT TO NEXT ENTRY
      BCT  R13,OTOP
      LA   R13,ER8
      B    ERROR4
OHIT   DS   0H
      L    R10,FPPCVD
      SPACE 3
*****
*       SET UP MOVE INSTRUCTION                           *
*****
      SPACE 3
      SR   R13,R13              CLEAR R13
      IC   R13,ILEN             INPUT LENGTH
      SR   R6,R13               POINT TO INPUT VALUE IN TABLE
      SH   R13,=H'1'           -1
      EX   R13,MVCINST
      BR   R14
      SPACE 3
*****
*       DEFINE ROUTINE                                     *
*****
      SPACE 3
DEFINE DS   0H
      MVC  FUNCT,=C'DEFN'
      LA   R5,BBB1
      B    CHKINP
BBB1  DS   0H
      SPACE 3
*****
*       UPDATE WORK AREA LENGTH IN FPIB                   *
*****
      MVC  FPBWKLN,=Y(WAEND-WA)
      SPACE 3

```

Figure 118 (Part 4 of 9). Field Procedure Example

```

*****
*      SET UP FIELD VALUE DESCRIPTOR      *
*****
      SPACE 3
      L    R10,FPPFVD          OUTPUT DESCRIPTOR
      MVC  FPDVTYPE,=Y(FPDVDTCHR)  FIXED CHARACTER
      MVI  FPDVLEN,X'00'
      AH   R10,=H'3'
      MVC  0(1,R10),OLEN
      BR   R14
      SPACE 3
*****
*      CHECK INPUT ROUTINE          RETURN R5      *
*****
      SPACE 3
CHKINP  DS   0H
      L    R8,FPPPV L
      L    R10,FPPCVD          INPUT DESCRIPTOR
      CLC  =Y(FPDVDTCHR),FPDVTYPE  FIXED CHARACTER ?
      BNE  CHKINPE1
      CLC  FPPVCNT,=H'1'          ONLY ONE PARAMETER ?
      BNE  CHKINPE2              NO, ERROR
      LA   R7,TBLHDR            POINT TO TABLE HEADER TABLE
LOOP1   DS   0H
      CLC  CODE,FPPVDS+8          IS VALUE IN TABLE
      BNE  CPEND                 NO, INCREMENT
      B    CINCL                 YES, A HIT
CPEND   DS   0H
      AH   R7,=H'8'              EACH TABLE ENTRY 8 BYTES
      CLI  CODE,X'FF'            END OF TABLE?
      BNE  LOOP1                 NO
      LA   R13,ER3
      B    ERROR8                YES, ERROR
CINCL   DS   0H
      CLC  ILEN,FPDVLEN+1        CHECK INPUT LENGTH
      BER  R5
      LA   R13,ER4
      B    ERROR4
CHKINPE1 DS  0H
      LA   R13,ER1
      B    ERROR4
CHKINPE2 DS  0H
      LA   R13,ER2

```

Figure 118 (Part 5 of 9). Field Procedure Example

```

ERROR8 DS 0H
MVC FPBRTNC,=AL2(FPBRC8)
B ERROR
ERROR4 DS 0H
MVC FPBRTNC,=AL2(FPBRC4)
ERROR DS 0H
MVC FPBRSNC,FUNCT
ST R13,FPBTOKP
BR R14
SPACE 3
*****
* CHECK OUTPUT DESCRIPTOR RETURN R5 *
*****
SPACE 3
CHKOUT DS 0H
L R10,FPPFVD FIELD DESCRIPTOR
CLC =Y(FPVDTCHR),FPVDTYPE FIXED CHARACTER ?
BNE CHKOUTE1
CLC OLEN,FPVDVLEN+1 CHECK OUTPUT LENGTH
BER R5
LA R13,ER6
B ERROR4
CHKOUTE1 DS 0H
LA R13,ER7
B ERROR4
SPACE 3
*****
* SET UP LOOKUP VARIABLE ROUTINE RETURN R5 *
*****
SPACE 3
SETLUP DS 0H
SR R4,R4 CLEAR R4
IC R4,ILEN INPUT LENGTH
ST R4,INCRLEN SAVE INPUT LENGTH
SR R4,R4 CLEAR R4
IC R4,OLEN OUTPUT LENGTH
A R4,INCRLEN ADD INPUT LENGTH
ST R4,INCRLEN STORE TABLE ENTRY LENGTH
SR R13,R13 CLEAR R13
IC R13,NENTR NUMBER OF ENTRIES
BR R5
SPACE 3

```

Figure 118 (Part 6 of 9). Field Procedure Example


```

*****
*          ERROR MESSAGES          *
*****
ER1      DC      CL40'INVALID COLUMN TYPE'
ER2      DC      CL40'INVALID NUMBER OF PARAMETERS'
ER3      DC      CL40'INVALID PARAMETER VALUE'
ER4      DC      CL40'INVALID COLUMN LENGTH'
ER5      DC      CL40'INVALID INPUT VALUE TO ENCODE'
ER6      DC      CL40'INVALID FIELD LENGTH'
ER7      DC      CL40'INVALID FIELD TYPE'
ER8      DC      CL40'INVALID FIELD VALUE TO DECODE'
          SPACE 3
*****
*          TABLE HEADER TABLE    *
*****
TBLHDR   DS      0F
*****
*          FIRST TABLE  CODE = 'A'  *
*****
          DC      C'A'                CODE
          DC      X'01'                INPUT LENGTH
          DC      X'01'                OUTPUT LENGTH
          DC      X'03'                NUMBER OF ENTRIES
          DC      A(TABA)              ADDRESS OF LOOKUP TABLE
*****
*          SECOND TABLE  CODE = 'B'  *
*****
          DC      C'B'                CODE
          DC      X'04'                INPUT LENGTH
          DC      X'01'                OUTPUT LENGTH
          DC      X'22'                NUMBER OF ENTRIES
          DC      A(TABB)              ADDRESS OF LOOKUP TABLE
*****
*          PUT ADDITIONAL TABLE HEADER ENTRIES HERE          *
*****
          SPACE 3
*****
*          END OF TABLE HEADERS          *
*****
          DC      X'FF'
          SPACE 3
TABA     DS      0H
          DC      C'H'                HIGH
          DC      C'7'
          DC      C'M'                MEDIUM
          DC      C'5'
          DC      C'L'                LOW
          DC      C'3'
          SPACE 3

```

Figure 118 (Part 7 of 9). Field Procedure Example

TABB	DS	0H	
	DC	C'AAA '	
	DC	X'F0'	240
	DC	C'AA+ '	
	DC	X'E6'	230
	DC	C'AA '	
	DC	X'DC'	220
	DC	C'AA- '	
	DC	X'D2'	210
	DC	C'A+ '	
	DC	X'C8'	200
	DC	C'A1 '	
	DC	X'BE'	190
	DC	C'A '	
	DC	X'B4'	180
	DC	C'A- '	
	DC	X'AA'	170
	DC	C'BBB+ '	
	DC	X'A0'	160
	DC	C'BBB '	
	DC	X'96'	150
	DC	C'BBB- '	
	DC	X'8C'	140
	DC	C'BB+ '	
	DC	X'82'	130
	DC	C'BB '	
	DC	X'78'	120
	DC	C'BB- '	
	DC	X'6E'	110
	DC	C'B+ '	
	DC	X'64'	100
	DC	C'B '	
	DC	X'5A'	90
	DC	C'B- '	
	DC	X'50'	80
	DC	C'CCC '	
	DC	X'46'	70
	DC	C'CC '	
	DC	X'3C'	60
	DC	C'C '	
	DC	X'32'	50
	DC	C'D '	
	DC	X'28'	40
	DC	C'NR '	
	DC	X'1E'	
	SPACE	3	

Figure 118 (Part 8 of 9). Field Procedure Example

```

*****
*           TABLE HEADER TABLE DSECT           *
*****
TBLHDRD  DSECT
CODE     DS     CL1
ILEN     DS     CL1
OLEN     DS     CL1
NENTR    DS     CL1
TABADDR  DS     A
          SPACE 3
*****
*           WORK AREA                             *
*****
          SPACE 3
WA        DSECT
SAVE13   DS     F
INCRLEN  DS     F
FUNCT    DS     CL4
WADW     DS     D
WAEND    DS     0H
          SPACE 3
          ARIBFPB
R0        EQU   0
R1        EQU   1
R2        EQU   2
R3        EQU   3
R4        EQU   4
R5        EQU   5
R6        EQU   6
R7        EQU   7
R8        EQU   8
R9        EQU   9
R10       EQU  10
R11       EQU  11
R12       EQU  12
R13       EQU  13
R14       EQU  14
R15       EQU  15
          END

```

Figure 118 (Part 9 of 9). Field Procedure Example

End of General-Use Programming Interface

Chapter 15. Using a DRDA Environment

A Distributed Relational Database Architecture (DRDA) environment provides full SQL client-server architecture for remote unit of work access to data that is distributed across different installations. The application requester and the application server do not have to be running with the same database manager.

Not all extended features are supported by the DRDA protocol. Refer to Appendix H, “DRDA Considerations” on page 499.

For detailed information on Distributed Relational Database Architecture, see the various manuals in the *Distributed Relational Database Architecture Library* manual, listed in the Bibliography.

The following two sections describe some of the benefits and added responsibilities respectively, that you should be aware of when using the DRDA protocol. Because each installation is different, these discussions cannot be inclusive.

Benefits of Using the DRDA Protocol

With the DRDA protocol, DB2 Server for VM users can use remote unit of work processing on non-DB2 Server for VM application servers. This allows access to data that could otherwise remain unavailable (that is, now application programs that were restricted to the data in DB2 Server for VM databases can now access data controlled by non-DB2 Server for VM application servers as well). In addition, OS/2, AIX, OS/400, MVS, or Microsoft Windows NT users can run applications that utilize DRDA remote unit of work processing or DRDA distributed unit of work processing to access data residing in DB2 Server for VM application servers.

To support this access, application programs can contain SQL statements that are specific to the target system, and both the DBS utility and ISQL can be run on non-DB2 Server for VM application servers. The SQL statements in these application programs can be static, dynamic, and extended dynamic, even if the target system does not support extended dynamic statements. In addition, portable packages can be loaded on non-DB2 Server for VM application servers.

Other than enabling access to non-DB2 Server for VM systems, the DRDA protocol provides additional benefits:

- To determine the status of connections in an environment that may have local and remote systems, you can use either the SHOW CONNECT operator command or the SQLQRY CMS immediate command.
- To aid in the diagnosis of errors, first failure data capture is automatically performed. IBM service can use the captured data for diagnosis, decreasing the probability of having to rerun applications to acquire data for diagnosis.
- Another aid in the diagnosis of errors is the LUWID support. The LUWID is a unique identifier associated with each application requester connection. It is composed of four parts: network id, LU name, LUW instance number, and LUW

sequence number. This provides additional information that may be required in problem diagnosis.

Added Responsibilities in Using the DRDA Protocol

Use of the DRDA protocol requires assuming extra responsibilities that are usually not required in a non-distributed environment.

Because the communications between database managers can be in different time zones or countries, some allowance must be made for scheduling and communication problems (particularly when different languages are involved).

The operation of applications may be similar, but the different platforms will require modifications. These modifications may require that as system administrator you become familiar with the terminology used on non-DB2 Server for VM database managers. In situations such as adding users, assigning resources, ascertaining the authorization schemes available, and performing diagnosis, the different terminology of the different database managers can lead to misunderstandings. Similarly, because communications software is involved, you may have to become familiar with communication terminology that may not be required in a non-distributed environment.

Applications that run in a DRDA environment also require attention. In some instances, they may have to be recoded to compensate for system-to-system processing differences. As an example, consider the differences between collating sequences on different database managers. Quite apart from the differences between the ASCII and EBCDIC collating sequences, differences can occur between EBCDIC collating sequences on two different database managers: the same character can appear in a different sequence because of the way in which a system processes information. If an application is not recoded to correct for this variability, the results generated by that application can be misleading.

Preparing to Implement DRDA

If you plan to use the DRDA code, you must prepare both the application requester and the application server to run in a distributed environment. This section provides a checklist of the required tasks. For detailed information on DRDA, see the *Distributed Relational Database Connectivity Guide*.

On the application requester:

- Increase the virtual storage (see Table 36 on page 444)
- Provide the following network information:
 - Define the local system
 - Define the remote systems
 - Supply information to the VTAM product and to AVS
 - Select RU sizes and pacing
 - Review the existing network definitions to ensure that the distributed database system does not adversely affect the existing network
 - Update the CMS communications directory

- Provide the following security controls:
 - Supply information to the VTAM product for the level of security required
- Specify the PROTOCOL parameter of the SQLINIT EXEC
- Preprocess the applications to be used
- Compile the applications.

On the application server:

- Increase the virtual storage (see Table 36 on page 444)
- Define DB2 Server for VM components in saved segments (optional), since the size of the resource adapter and RDS components will increase.
- If the application server is to be used for distributed unit of work activity, install a CRR recover server. For more information, see the *VM/ESA: Installation Guide* and the *VM/ESA: CMS File Pool Planning, Administration, and Operation* manuals.
- Provide the following network information:
 - Define the application server to the VTAM product and to AVS
 - Review the existing network definitions to ensure that the distributed database system will not adversely affect the existing network
- Provide the following security controls:
 - Either use come-from checking so that the application server accepts user IDs from specific locations only (and translates a user ID to a different user ID when necessary), or ensure that all user IDs are unique within the SNA network
 - Ensure that the application server provides a security manager product to support the required security features
- Specify the PROTOCOL parameter of the SQLSTART EXEC.

Installing and Removing the DRDA Code

Installing the DRDA code is an optional customization step. You can install it:

- Immediately after installing or migrating the base code
- At a later date, whenever it is required
- On *both* the application server and the application requester
- On *either* the application server or the application requester

You can later remove the DRDA code if it is no longer required.

The DRDA code is implemented using the PROTOCOL parameter. On the application server, this parameter is on the SQLSTART EXEC; on the application requester, it is on the SQLINIT EXEC.

When the application server is started with PROTOCOL=AUTO, access from DB2 Server for VM *and* non-DB2 Server for VM application requesters is allowed. When the application requester specifies PROTOCOL(AUTO) or PROTOCOL(DRDA), connections to DB2 Server for VM *and* non-DB2 Server for VM application servers are allowed.

When the application server is started with `PROTOCOL=AUTO` and `SYNCPNT=Y`, distributed unit of work support is enabled where servers can participate in multiple-site read multiple-site update logical units of work. This requires DB2 Server for VM to interface with VMCRR. If `SYNCPNT=N` is specified, then the DB2 Server for VM server does not interface with VMCRR and the server is restricted to multiple-site read single-site update units of work. When `SYNCPNT=N`, the DB2 Server for VM server may be one of the multiple sites where data is only read, or it may be the single site where data is updated.

Note: If `SYNCPNT=Y` is specified but a CRR Recovery server has not been installed, distributed unit of work is not possible. Error messages are issued and DB2 Server for VM runs as if the `SYNCPNT` startup parameter had been set to N.

Do not install the DRDA code unless it is specifically required, as the increased processing required for distributed conversations requires a significant amount of storage. For details on virtual storage requirements, see Appendix A, "Virtual and Real Storage Requirements" on page 443.

Steps to Install or Remove the DRDA Code

To install (or remove) the DRDA code, perform the following steps on the database machine and installation user ID:

1. Log on to the database machine (SQLMACH).
2. Stop the application server using your normal operating procedures.
3. Ensure that the database machine production disk and service disk are linked in write mode. If not, enter:

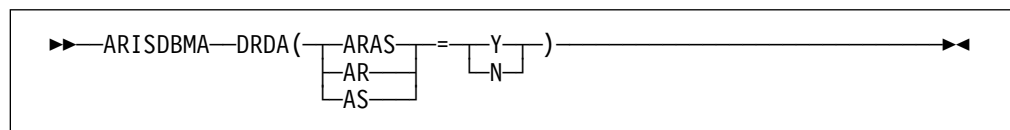
```
LINK machid 195 195 W
LINK machid 193 193 W
```

4. Access the production disk with file mode Q and the service disk with file mode V.

```
ACCESS 195 Q
ACCESS 193 V
```

If you are using SFS directories instead of minidisks, access them with file modes Q and V.

5. Run the ARISDBMA EXEC to identify whether you want application server or application requester DRDA code installed on your production disk. Its syntax is:



Specify any of the following combinations of parameters:

ARAS=Y Install the DRDA code for both the application server and application requester (this is the default).

ARAS=N Remove the DRDA code from both the application server and application requester.

- AR=Y** Install the DRDA code for the application requester.
- AR=N** Remove the DRDA code from the application requester.
- AS=Y** Install the DRDA code for the application server.
- AS=N** Remove the DRDA code from the application server.

For example, to identify that you want to install the DRDA code on the application server and remove it from the application requester, type:

```
ARISDBMA DRDA(AS=Y)
ARISDBMA DRDA(AR=N)
```

6. Log off the database machine.

7. Log on to the DB2 Server for VM installation user ID, **5648A70S**

You should have read access to the VMSES/E code (MAINT 5E5 disk) and read/write access to the Software Inventory disk (MAINT 51D) or SFS directory.

8. Establish the access order.

```
vmfsetup 5648A70S {DB2VM|DB2VMSFS}
```

5648A70S is the PPF that was shipped with the product. If you have your own PPF override, substitute that name for 5648A70S shown in this command. You also need to substitute your PPF name in the VMSES/E commands in any subsequent steps.

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

9. Rebuild DB2 Server for VM ARISQLLD LOADLIB.

a. Rebuild the ARISQLLD LOADLIB.

```
vmfbld ppf 5648A70S {DB2VM | DB2VMSFS} ARIBLLLD (all
vmfview build
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

ARIBLLLD is the name of the VMSES/E build list used to build the ARISQLLD LOADLIB.

Review the build message log (\$VMFBLD \$MSGLOG). If necessary, correct any problems before you continue. Use the PF2 key, ALL, to review all of the messages.

Notes:

1) The following message is normal if you are **NOT** running DB2 Server for VM with the DB2 Data Spaces Support:

```
VMFLB2074I Part xxxxxxx TXT in object ARISQLDS
in build list ARIBLLLD
EXEC will be ignored
```

2) The following message is normal if you are **NOT** running DB2 Server for VM with the DRDA application server support:

```
VMFLB2074I Part xxxxxxx TXT in object ARIXRDS
in build list ARIBLLLD
EXEC will be ignored
```

3) The following message is normal if you are **NOT** running DB2 Server for VM with the DRDA application requester support:

```
VMFLB2074I Part xxxxxx TXT in object ARIRVMRM
in build list ARIBLLD
EXEC will be ignored
```

b. Build the related files.

```
vmfbld ppf 5648A70S {DB2VM | DB2VMSFS} (serviced
vmfview build
```

Use DB2VM for installing on minidisks or DB2VMSFS for installing in Shared File System directories.

Review the build message log (\$VMFBLD \$MSGLOG). If necessary, correct any problems before you continue.

10. Link and access the database machine user ID production and service disks or SFS directories.

```
link SQLMACH 195 295 MR
acc 295 l
link SQLMACH 193 293 MR
acc 293 m
```

You will be prompted for the password to the disks.

Substitute your minidisk addresses, if different. Substitute in the appropriate SFS directory names. You also need to substitute your minidisk address and SFS directory names in the VMSES/E commands in any subsequent steps.

11. Copy the new ARISQLLD LOADLIB to SQLMACH's production and service disk or directory.

a. If installing on minidisks, enter the following commands:

```
access 195 i
vmfcopy arisqlll L* i = = l (prodid 5648A70S%DB2VM olddate replace
access 193 j
vmfcopy arisqlll L* j = = m (prodid 5648A70S%DB2VM olddate replace
```

The VMFCOPY command updates the VMSES PARTCAT file on the production disk (195) and the service disk (193).

b. If installing using Shared File System, enter the following commands:

```
access 5648A70S.sql.production i
access SQLMACH.sql.production l
vmfcopy arisqlll L* i = = l (prodid 5648A70S%DB2VM olddate replace
access 5648A70S.sql.service j
access SQLMACH.sql.service m
vmfcopy arisqlll L* j = = m (prodid 5648A70S%DB2VM olddate replace
```

The VMFCOPY command updates the VMSES PARTCAT file.

12. If you previously stored any DB2 Server for VM components in saved segments, you should resave them (refer to "Step 11. Prepare to Build the DB2 Server for VM Segments" on page 195 through "Step 15. Create a Bootstrap Package" on page 198 in Chapter 8, "Saved Segments" on page 181). In "Step 14. Build the DB2 Server for VM Segments" on page 197 on the VMFBLD command, use the "(add)" option rather than the "(serviced)" option.

The sizes of the resource adapter and RDS increase with the DRDA code installed. If you need to increase the size of any of the saved segments, you must run VMFSGMAP to redefine the segment (see "Step 4. Prepare to Add DB2 Server for VM Segment Definitions" on page 190).

13. Log off the installation user ID.
14. You must define a CMS communications directory to access a remote application server through the VTAM product. For more information, see "Setting Up the CMS Communications Directory" on page 12.
15. Log on the database machine and restart the application server in multiple user mode with the required PROTOCOL parameter.

Using the DBS Utility and ISQL on Other Application Servers

For a user to be able to use the DBS utility on a non-DB2 Server for VM application server, you must first preprocess the DBS utility package on the non-DB2 Server for VM application server and then create the table SQLDBA.DBSOPTIONS on that application server. This is done by the DB2 Server for VM application requester. You must then obtain the necessary program bind and table creation privileges for your authorization-id on the target application server.

Note: If the target application server does not support the ERROR option on the preprocessing, you must create the DB2 Server for VM system catalog tables on the target application server for the preprocessing to work. The database managers that do not support the ERROR option (such as the common server database managers) generally supply a command file that creates the necessary table definitions. For example, DB2 for OS/2 supplies the files SQLDBSU.CMD and ISQL.CMD, which create the tables that must exist to create the DBS Utility and ISQL packages, respectively. Similarly, DB2 for NT supplies SQLDBSU.BAT and ISQL.BAT.

Creating the DBS Utility Package

If the target application server does not support the ERROR option, create the necessary table definitions. This must be done on the target application server, which should supply a job called SQLDBSU.CMD or SQLDBSU.BAT to create the tables. Run the supplied job.

Do the following from a DB2 Server for VM application requester:

1. To establish the non-DB2 Server for VM application server as the default application server, run the SQLINIT EXEC against it.

2. To link to the database machine's service disk, enter:

```
LINK machid 193 193 RR
```

3. To access the service disk, enter:

```
ACC 193 V
```

4. To preprocess the DBS utility, enter:

```
SQLPREP ASM PP (PREP=SQLDBA.ARIDSQ, BLOCK, ISOL(CS), NOPR, ERROR, NOPU,  
CTOKEN(NO)) IN (ARIDSQLP MACRO V)
```

Omit the ERROR option on the SQLPREP statement if the target application server does not support it.

5. If you ran a command file to create the table definitions necessary for the preprocessing, the DBSOPTIONS table should have been created for you. If this table does not exist, enter the following DBS Utility commands:

```

SET ERRORMODE CONTINUE;

CREATE TABLE SQLDBA.DBSOPTIONS
  (SQLOPTION VARCHAR (18) NOT NULL,
   VALUE     VARCHAR (18) NOT NULL);

CREATE UNIQUE INDEX SQLDBA.DBSINDEX
  ON SQLDBA.DBSOPTIONS (SQLOPTION,VALUE);

INSERT INTO SQLDBA.DBSOPTIONS
  VALUES ('RELEASE','6.1.0');

COMMIT WORK;

```

You must now obtain the necessary program bind and table creation privileges for your authorization-id on the target application server.

Creating the ISQL Package

For a user to be able to make ISQL requests against a non-DB2 Server for VM application server, you must load the ISQL package on that application server.

Before you can load ISQL on a non-DB2 Server for VM application server, you must first preprocess the DBS utility on the non-DB2 Server for VM application server. When the DBS utility is preprocessed on the non-DB2 Server for VM application server, ensure that you have the necessary program bind and table creation privileges for your authorization-id on the target application server.

If the target application server does not support the ERROR option, run the ISQL.CMD or ISQL.BAT file. This file is supplied by the target server.

Do the following from a DB2 Server for VM application requester to load ISQL:

1. Run the SQLINIT EXEC to establish the non-DB2 Server for VM application server as the default application server.
2. To link to the database machine's service disk, enter:


```
LINK machid 193 193 RR
```
3. To access the service disk, enter:


```
ACC 193 V
```
4. Issue the following CMS command:


```
FILEDEF ARIISQLM DISK ARIISQLM MACRO V
```
5. Issue the following DBS utility command to reload ISQL:


```
RELOAD PACKAGE (SQLDBA.ARIISQL) REPLACE KEEP INFILE (ARIISQLM);
```
6. Create the table SQLDBA.ROUTINE, and any other *userid*.ROUTINE tables that you want.

For the CREATE TABLE statement that you use to create SQLDBA.ROUTINE, see the *DB2 Server for VSE & VM Interactive SQL Guide and Reference manual*.

Types of Access to Distributed Data

Two types of access to data in distributed relational database systems are currently available. They are *remote unit of work* and *distributed unit of work*.

Remote Unit of Work

Remote unit of work, implemented in SQL/DS V3.3 (for VM) and in SQL/DS 3.4 (for VSE), allows a user or an application to read or update data at one remote location per unit of work. With remote unit of work, you can have many SQL statements within a unit of work. You can access one database management system with each SQL statement, and you can access one database management system within a unit of work.

Consider a banking example. Using remote unit of work, you can transfer funds from a savings account table to a checking account table, if both tables are at the same remote location. Figure 119 shows how the application first requests an update to the savings account table (1) and then requests an update to the checking account table (2).

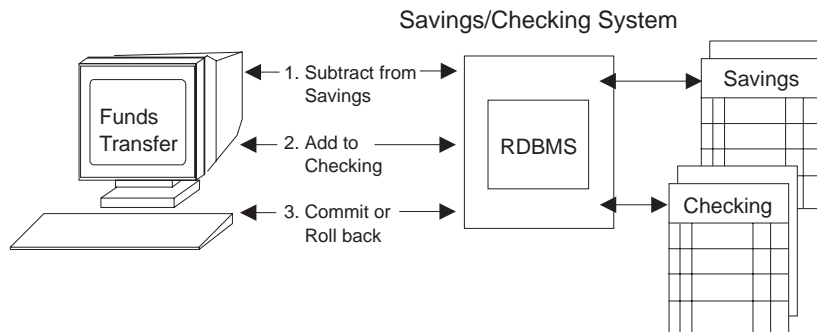


Figure 119. Remote unit of work

If both requests are processed successfully, the application can direct the database management system to commit both updates (3). If either request is not processed successfully, the updates are rolled back, leaving both tables as they were before the transaction began. This ensures that requests are neither lost nor duplicated.

The Distributed Two-Phase Commit Process

Distributed unit of work lets a user or application program read or update data at multiple locations within a single unit of work. With distributed unit of work, you can:

- Have many SQL statements within a unit of work
- Access one database management system with each SQL statement
- Access many database management systems within a unit of work.

Using the banking example (see Figure 120 on page 420), imagine that the savings account table and the checking account table are on two different computer systems. Distributed unit of work processing permits an application to debit the savings account (1), credit the checking account (2), and either commit or roll back the operations in both computer systems (3), treating all of the changes as a single transaction, or unit of work.

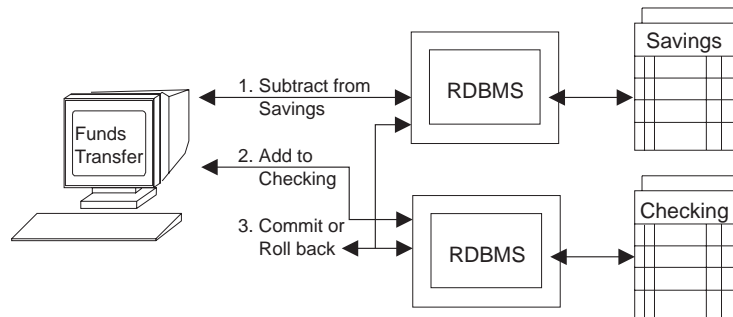


Figure 120. Distributed unit of work

Commit and rollback are coordinated at all locations so that if a failure occurs anywhere in the system, data integrity is preserved. If there was a failure in the middle of the banking transaction just described, and commit or rollback was not coordinated, the savings account could be debited money and the checking account might not be credited the money. This costly error is avoided by the coordination of commit and rollback, or *two-phase commit processing*.

Two-Phase Commit Processing

Distributed unit of work is a coordinated approach involving two phases. This coordination is done by a *sync point manager*. DB2 Server for VM uses VMCR as its sync point manager. A sync point manager maintains consistency in changes which are made to protected resources. The primary functions of a sync point manager include, but are not limited to, the following:

1. Keeping track of and logging LUW state information
2. Keeping track of and logging all local protected resource manager (PRM) names that are involved with a logical unit of work
3. Coordinating the COMMIT and ROLLBACK of all local PRMs
4. Initiating resynchronization protocols for any logical unit of work that may be in the in-doubt state because of a system or communications failure.

A sync point manager is required wherever resources may be updated. This requires that sync point managers at each distributed location communicate with one another using architected protocols. These protocols are fully discussed in the *SNA LU 6.2 Reference: Peer Protocols* manual.

For a full explanation of what two-phase commit is, see the following manuals:

- *IBM Systems Network Architecture, Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM Systems Network Architecture, Logical Unit 6.2 Reference: Peer Protocols*
- *IBM Distributed Relational Database Architecture Reference*
- *Distributed Data Management (DDM) List of Terms*.

Using the Two-Phase Commit Protocol

An example of a two-phase commit protocol sequence is shown in Figure 121 on page 422. SNA LU 6.2 functions provide so many capabilities that it is impossible to show all the possible sequences. Notes describing key points in the sequence follow the sequence diagram.

The following assumptions have been made for the example:

- A conversation has been successfully established between the Source Server and the target communications manager (TCM) using a protected conversation.
- No error situation occurs.

For example:

- The "Source Server" could be DDCS Multi-User Gateway V2.3.1. In this case, the "SYNCPNTMGR" would be function included with DDCS. Also, the "SNA LU 6.2" function could be provided by Communications Server for OS/2 Version 4.
- The "Target Server" would be DB2 Server for VSE. The "TCM" is the communication function of DB2 Server for VM. The "SYNCPNTMGR" would be VM/CRR. The "Other Protected Managers" would be the database manager function of DB2 Server for VM.

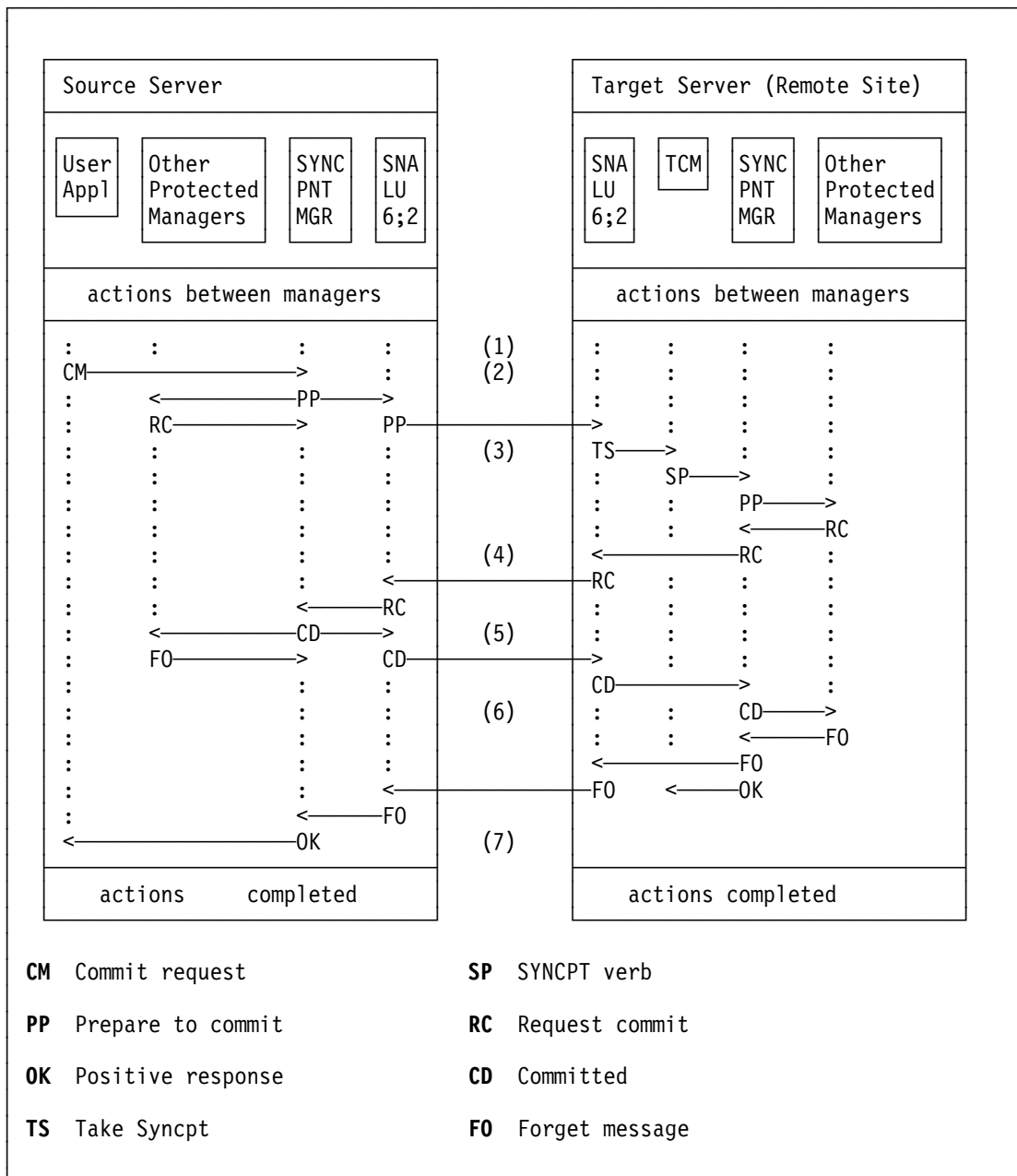


Figure 121. Successful Two-Phase Commit

Figure Notes:

- (1) The Target Communications Manager (TCM) issues a RECEIVE_AND_WAIT APPC verb to receive the next SQL Request from the Source Server.
- (2) The source application program requests the SYNC PNT MGR to commit the logical unit of work (LUW). The source SYNC PNT MGR notifies the SNA LU 6.2 communications facilities to prepare to commit and notifies the source database (and other protected resource managers registered with the SYNC PNT MGR) to prepare to commit. The source communications facility sends the SNA LU 6.2 prepare message to the target system. The local protected

resource managers respond to the source SYNCNTMGR with the "Request Commit" message.

- (3) On the target system, the RECEIVE_AND_WAIT verb is completed and the WHAT_RECEIVED parameter is set to TAKE_SYNCPT.
The TCM issues a SYNCPT verb to the target SYNCNTMGR which begins the commit processing. The SYNCNTMGR prepares the protected resources to commit.
- (4) The SYNCNTMGR sends the SNA LU 6.2 request commit message to the source system.
- (5) The source SYNCNTMGR collects the request commit messages from the SNA LU 6.2 communications facilities and the other protected resource managers. The source SYNCNTMGR then commits the logical unit of work by requesting that all of the resources commit. This causes an SNA LU 6.2 committed message to be sent to the target system.
- (6) The target SYNCNTMGR requests that the local resources commit the logical unit of work and causes an SNA LU 6.2 forget message to be sent to the source system. In addition, the target SYNCNTMGR posts a positive response to the TCM for the SYNCPT verb issued in note (3).
- (7) When the source SYNCNTMGR receives the "FO" responses from the protected resource managers, a positive response to the commit is given to the application program.

In the VM environment, DB2 Server for VM uses VM/ESA Coordinated Resource Recovery (VMCRR) as its sync point manager. VMCRR uses the Callable Services Library (CSL) routines to customize the sync point manager function. The environment can be shown as follows:

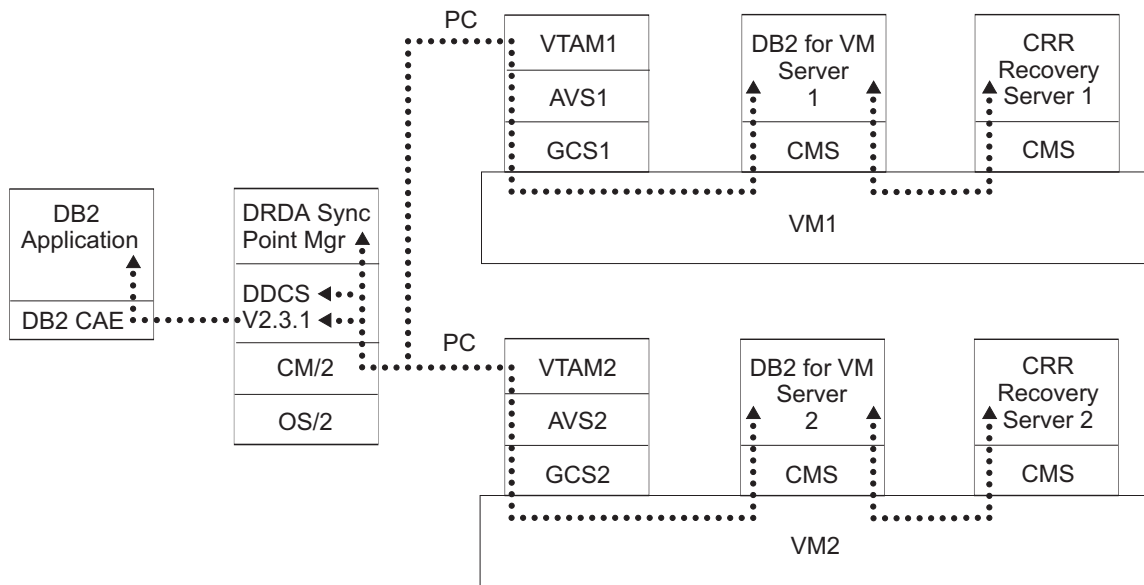


Figure 122. DB2 Server for VM Distributed Unit of Work Environment.

This diagram shows how a workstation application might use DDCS for OS2 V2.3.1 to execute a distributed unit of work between 2 DB2 Server for VM servers on

different VM/ESA systems. Notice that DDCS registers itself with its own sync point manager. DDCS then establishes protected conversations with DB2 Server for VM servers 1 and 2. Each DB2 Server for VM server uses the CRR Recovery server installed on its VM system to perform any sync point logging and resynchronization activity. Notice also that each DB2 Server for VM server registers itself with the CRR Recovery server on its system.

CAE	DB2 Client Application Enabler
DDCS	Distributed Database Connection Services (V2.3.1)
CS/2	Communication Server for OS/2 (V4)
OS/2	OS/2 Warp
PC	Protected Conversation

Operator Commands

The following DB2 Server for VM operator commands can be used to manage in-doubt LUWs:

SHOW ACTIVE	Displays the status of active agent structures ¹
SHOW CONNECT	Displays the status of all users or selected users connected to the application server.
FORCE COMMIT	Heuristically forces an in-doubt LUW to COMMIT
FORCE ROLLBACK	Heuristically forces an LUW to ROLLBACK
SHOW INDOUBT	Displays the status of all DRDA2 distributed units of work that are: <ul style="list-style-type: none">• currently in-doubt• were heuristically committed or rolled back but RESYNC has not yet been performed nor has RESET INDOUBT been performed• RESYNC failed for some in-doubt unit of work, because the status of the LUW was the opposite of what RESYNC required. (for example, RESYNC required that the unit of work be COMMITTED, but it had been heuristically ROLLED BACK). At the same time, RESET INDOUBT has not been performed.
RESET INDOUBT	Causes a heuristically committed or rolled back unit of work to be forgotten by the database, (that is, causes a <i>forget</i> log record to be written.)

CRR Operator Commands

In addition to the DB2 Server for VM operator commands, the following VM/ESA CRR operator command may be used to manage activity at the CRR operator console:

CRR ERASE LU	Erases specified LU name and TPN entries from the CRR log name table
---------------------	--

¹ An agent is the internal database manager representation of an active user

CRR ERASE LUWID	Erases CRR log records for a specified LUWID instance, which prevents any further CRR recovery server activity on this LUWID instance
CRR QUERY LOG	Displays the status of the CRR log minidisks
CRR QUERY LOGTABLE	Displays LU names and TPNs in the CRR log name table
CRR QUERY LU	Displays status of logical units of work known to this CRR recovery server and associated with the specified LU name
CRR QUERY LUWID	Displays status of sync point processing and resynchronization processing for an LUWID instance known to this CRR recovery server
CRR RESUME	Restarts the automatic periodic retry of resynchronization for a specified LUWID that was suspended by the CRR SUSPEND command and also bypasses the timed wait interval
CRR RESYNC	Provides a heuristic response for an unavailable protected resource or protected conversation so resynchronization can continue
CRR SUSPEND	Stops the automatic periodic retry of resynchronization for a specified LUWID until the CRR operator enters the CRR RESUME command

These CRR commands are discussed in the *VM/ESA CMS File Pool Planning, Administration, and Operation* manual.

Resynchronization

Resynchronization occurs if two-phase commit processing is interrupted by a resource failure. A resource failure may be caused by a node failure, a session failure, a program failure or other problems by a protected resource manager. The resource failure may be between a sync point manager and local resource managers or sync point manager and remote resource managers.

Resynchronization is conducted independently for each failed protected resource for which it is required. Resynchronization has the following purposes:

- To place distributed resources in consistent states, if possible; if not possible, to notify the operator at the LU that detected the damage and at the LU of the root of the sync point tree. The LU for DB2 Server for VM is AVS.
- To unlock locked resources in order to free them for other uses
- To update the log showing that no more sync point work is needed for that protected resource, for that LUW.

Resync When Partner is Not Active

After an LU failure, it is possible that the partner that is responsible for resync is unable to establish the resync conversation because the failed LU has not been restarted. The responsible LU retries the resync at implementation-defined intervals.

In order to reduce the delay for resynchronization after an LU is restarted, the partner LU may signal to the resync initiator that it is available by sending an

Exchange Log Names GDS variable that is **not** accompanied by a Compare States GDS variable.² Once the responsible LU has received this signal that the failed LU is active, it can initiate resync, sending the Exchange Log Names and Compare States GDS variables.

Sending the Exchange Log Names GDS variable as a signal of LU availability need be done only once, no matter how many protected conversations require resynchronization between the two LUs. Also, if the LU that becomes available is responsible for initiating resync for some conversations, it need not send another Exchange Log Names GDS variable as a signal that the LU is available, since the partner SPM can infer that a partner is available from the other resyncs the partner SPM initiates.

Resynchronization Initialization

To notify the CRR recovery server of its readiness to accept resynchronization communications, a DB2 Server for VM server performs *Resynchronization Initialization* with the CRR recovery server when it starts up³ in multiple user mode. This involves sending an exchange log names request to the CRR recovery server before the first sync point is processed. The CRR recovery server sends an exchange log names reply. Then DB2 Server for VM and the recovery server save each other's log name, LU name and TPN to use in later validations. The current values for these can be determined using the database manager's SHOW CRR LOGNAMES command. For more information, see the *DB2 Server for VSE & VM Operation* manual.

As part of the exchange log names request, DB2 Server for VM sends a *log name*. The DB2 Server for VM log name is the concatenation of the following information (with blank characters stripped off):

- SNA NETID
- The database manager's VM node ID
- The database manager's RESID.

The database manager also sends a *log status* in the exchange log names request. Log status can have one of the following values:

warm	When the database manager starts up, it invokes the DMSGETRS CSL routine to determine the current TPN of the CRR recovery server. If the TPN value returned by the DMSGETRS routine matches the TPN value in the database manager's log, the log status is warm.
cold	When the database manager starts up, if it determines that the current TPN value of the CRR recovery server does NOT match the value of the CRR recovery server's TPN as stored in the database manager's log, the log status is cold.

The database manager also send a TPN value in the exchange log names request. The TPN value is the RESID for the database manager.

² The partner can tell that a Compare States GDS variable is not present because SPM's RECEIVE_AND_WAIT verb will complete with a WHAT_RECEIVED of SEND rather than DATA_COMPLETE.

³ **Note** - this is *only* done when the database is initializing. Once database initialization has completed, resynchronization initialization is not performed again until the database is brought down and then restarted.

The CRR recovery server receives the exchange log names request, processes it, and returns an Exchange Log Names reply back to the database manager. This reply also contains a log status, which can have one of the following values:

- warm** The CRR recovery server compares the TPN value that DB2 Server for VM sent in the exchange log names request with the TPN value stored in the CRR recover server's log. If the values match, then the log status is warm.
- cold** If the TPN values sent by DB2 Server for VM in the exchange log names request does not match the TPN value stored in the CRR recovery server's log, the log status is cold.

The database manager receives the exchange log names reply from the CRR recovery server. The resulting action for DB2 Server for VM is described in the following table:

Table 32 (Page 1 of 2). Actions by DB2 Server for VM on Exchange Log Names reply.. When DB2 Server for VM receives an Exchange Log Names reply from the VM CRR recovery server, the database manager needs to do certain actions. This table summarizes these required actions.

CRR Recovery Server's Log Status	Reply from CRR	DB2 Server for VM's log status	DB2 Server for VM's action
COLD or WARM	NORMAL	COLD	The database manager saves or updates the recovery server's locally known LU name, fully qualified LU name, TPN, and log name in its log and sends an explicit APPC confirmation to the recovery server.
COLD	NORMAL	WARM	<p>The database manager checks the work unit records in its log that relate to the recovery server.</p> <ul style="list-style-type: none"> • If no DRDA2 in-doubt logical units of work are found, DB2 Server for VM updates the recovery server's log name saved in the log and sends an explicit APPC confirmation to the recovery server. • If DRDA2 in-doubt logical units of work are found, DB2 Server for VM issues the following messages (the first one being equivalent to CMS message 3373E) on its operator console and does a deallocate (abend): <pre style="margin-left: 40px;">ARI0177E CRR recovery server at TPN <i>tpn</i> has provided a new log name resulting from a cold start. Some LUWID(s) cannot be automatically resolved by resynchronization. ARI0176I The SYNCPT parameter has been reset to N.</pre> <p>The database manager's participation in sync points must be delayed until the error condition is resolved. The DB2 Server for VM operator should contact the recovery server operator to determine the reason for the status mismatch. The operator might have to manually force some units of work using the FORCE operator command.</p>

Table 32 (Page 2 of 2). Actions by DB2 Server for VM on Exchange Log Names reply.. When DB2 Server for VM receives an Exchange Log Names reply from the VMCR recovery server, the database manager needs to do certain actions. This table summarizes these required actions.

CRR Recovery Server's Log Status	Reply from CRR	DB2 Server for VM's log status	DB2 Server for VM's action
WARM	NORMAL	WARM	<p>The database manager compares the recovery server's log name sent in the reply with the name in DB2 Server for VM's log:</p> <ul style="list-style-type: none"> • If the names match, DB2 Server for VM sends an explicit APPC confirmation to the recovery server. • If the log names do not match, DB2 Server for VM issues the following messages (the first one being equivalent to CMS message 3372E) on its operator console and does a deallocate (abend): <pre> ARI0178E An Exchange Log Name's Reply sent by CRR recovery server at TPN tpn contained a log name which does not match the current {database manager CRR recovery server} log name. Log name in Reply: log_name Current Log name: log_name ARI0176I The SYNCNT parameter has been reset to N. </pre> <p>The database manager's participation in sync points must be delayed until the error condition is resolved. The DB2 Server for VM operator should contact the recovery server operator to determine the reason for the log name mismatch. The database may have been restored from an archive which resulted in the log name mismatch, or the recovery server might be using the wrong log, and should be restarted with the correct log. If the correct log cannot be supplied, both partners must be coldstarted. Note that the "RESET CRR LOGNAMES" operator command may be used to reset the CRR recovery server's luname, tpn and log name and force an DB2 Server for VM log status of cold.</p>
WARM	ABNORMAL	COLD or WARM	<p>The database manager issues the following messages (the first one being equivalent to CMS message 3371E) on its operator console and does a deallocate (abend):</p> <pre> ARI0179E An Exchange Log Name's Reply sent by CRR recovery server at TPN tpn contained an error status. ARI0176I The SYNCNT parameter has been reset to N. </pre> <p>The SYNCPOINT parameter value is set to N because the database manager's participation in sync points must be delayed until the error condition is resolved. The DB2 Server for VM operator should contact the recovery server operator to determine the reason for the error.</p> <p>If the problem is a log name mismatch, one of the partners might be using the wrong log, and should be restarted with the correct log. If the correct log cannot be supplied, both partners must be coldstarted. Note that the "RESET CRR LOGNAMES" operator command may be used to reset the CRR recovery server's luname, tpn and log name and force an DB2 Server for VM log status of cold.</p>

Resynchronization Recovery

The CRR recovery server initiates the *resynchronization recovery* function to ensure consistent completion of the sync point by all registered resources for which data was logged. Using information stored in its log, the CRR recovery server determines which resources managers (for example, DB2 Server for VM) should be included in the recovery and allocates APPC conversations with them.

To allocate an APPC conversation with DB2 Server for VM, the CRR recovery server uses information that DB2 Server for VM provided when it registered with the SPM (using the CSL routine DMSREG).

The resynchronization recovery transaction between the CRR recovery server and the database manager consists of two functions:

- **Exchanging log names**

The CRR recovery server initiates this exchange with DB2 Server for VM to ensure that the data they saved from resynchronization initialization are still valid.

- **Comparing states**

The CRR recovery server initiates this exchange with DB2 Server for VM to compare the state of the CRR logical unit of work with the state of the database manager's logical unit of work.

During resynchronization recovery, DB2 Server for VM receives the Exchange Log Names and Compare states requests. First, the database manager processes the Exchange Log Names request as described in the following figure.

Table 33. Actions by DB2 Server for VM on Exchange Log Names request.. When the database manager receives an Exchange log Names request from the VMCR recovery server, it needs to do certain actions in order to formulate a reply. This table summarizes these required actions.

CRR Recovery Server's Log Status	DB2 Server for VM's log status	DB2 Server for VM's actions
WARM	COLD	<p>The database manager holds the recovery server's log name from the request but does not update its own log or process the compare states request. It then sends an Exchange Log Names reply to the recovery server indicating cold log status and normal completion of the request. DB2 Server for VM waits for indication of a deallocate (abend server, then does a deallocate (normal).</p> <p>Note that if the <i>RESET CRR LOGNAMES</i> command is issued, then the log status at the database will be COLD.</p>
WARM	WARM	<p>The database manager compares the recovery server's log name in the request with the name that is saved in its log. DB2 Server for VM also validates its own log name specified in the request.</p> <ul style="list-style-type: none"> • If the log names match, DB2 Server for VM formulates (but doe Exchange Log Names reply indicating normal completion of the request. It then processes the compare states request. After DB2 Server for VM completes the Compare states processi replies in the same buffer. • If the log names do not match, DB2 Server for VM issues the f (equivalent to CMS message 3372E) on its operator console and sends an Exchange Log Names reply to the recovery server indicating abnormal completion of the request: <pre data-bbox="613 1060 1242 1192"> ARI0178E An Exchange Log Name's Request sent by CRR recovery server at TPN tpn contained a log name which does not match the current {database manager CRR recovery server} log name. Log name in Request: log name Current Log name: log name </pre> <p>The database manager does not process the compare states request. It waits for indication of a deallocate (abend) by the recovery server, then does a deallocate (normal).</p> <p>The DB2 Server for VM operator should contact the recovery server operator to determine the reason for the log name mismatch. A couple of possible reasons are:</p> <ul style="list-style-type: none"> – The database manager's NETID, LUNAME or TPN may have changed, which resulted in a different log name. – An archive from another DB2 Server for VM database may have b in this database. <p>If this is the case then the database must be coldstarted. Note that the "RESET CRR LOGNAMES" operator command may be used to reset the CRR recovery server's luname, tpn and log name and force a DB2 Server for VM log status of cold. If the recovery server is using the wrong log and cannot locate the correct log, DB2 Server for VM might have to manually force some units of</p>

Note:

The database manager will determine its log status by comparing the CRR Recovery Server's LUNAME and TPN passed in the request with that which was recorded in its log. If there is any mismatch then the log status is deemed to be **COLD**.

If the log name exchange was satisfactory, the database manager then processes the Compare States request as shown in Table 34 on page 431.

<p><i>Table 34 (Page 1 of 2). Actions by DB2 Server for VM on Compare States request. When the database manager receives a compare states request from the VMCRR recovery server, it needs to do certain actions depending on the state of the LUW at DB2 Server for VM and at the recovery server. This table summarizes these required actions.</i></p>		
LUW state at DB2 Server for VM	LUWID state sent by CRR recovery server	
	Backout	Committed
LUWID Not found	Send normal completion reply indicating Backout state. DB2 Server for VM notifies operator with message: ARI0183E The Sync Point Manager has asked to <u>ROLLBACK</u> this LUW but the database manager has no memory of it.	Send normal completion reply indicating Backout state. DB2 Server for VM notifies operator with message: ARI0183E The Sync Point Manager has asked to <u>COMMIT</u> this LUW but the database manager has no memory of it.
Indoubt (Prepared)	Drive backout of resource and send normal completion reply indicating Backout state. DB2 Server for VM indicates backout to the operator with the message: ARI0230I FORCE ROLLBACK with disable scheduled for agent <i>user id</i> because of resynchronization recovery.	Drive commit of resource and send normal completion reply indicating Committed state. DB2 Server for VM indicates committed to the operator with the message: ARI0230I FORCE COMMIT with disable scheduled for agent <i>user id</i> because of resynchronization recovery.
Heuristic Backout	Send normal completion reply indicating Heuristic Backout State	Send normal completion reply indicating Heuristic Backout State. DB2 Server for VM notifies operator with message: ARI0184A The Sync Point Manager has asked to <u>COMMIT</u> this LUW but the FORCE command was previously used to <u>ROLLBACK</u> it. In this case, the LUW will still appear when the SHOW INDOUBT command is executed. The LUW must be cleared using the RESET INDOUBT command. In addition, manual intervention is necessary to ensure that the LUW is in a consistent state at all sites where the LUW has been distributed. This may require intervention at this database manager, or possibly at another database manager. Manual intervention could mean manually fixing the data or possibly restoring an archive.

Table 34 (Page 2 of 2). Actions by DB2 Server for VM on Compare States request. When the database manager receives a compare states request from the VM-CRR recovery server, it needs to do certain actions depending on the state of the LUW at DB2 Server for VM and at the recovery server. This table summarizes these required actions.

LUW state at DB2 Server for VM	LUWID state sent by CRR recovery server	
	Backout	Committed
Heuristic Committed	<p>Send normal completion reply indicating Heuristic Committed state. DB2 Server for VM notifies operator with message:</p> <p>ARI0184A The Sync Point Manager has asked to <u>ROLLBACK</u> this LUW but the <u>FORCE</u> command was previously used to <u>COMMIT</u> it.</p> <p>In this case, the LUW will still appear when the SHOW INDOUBT command is executed. The LUW must be cleared using the RESET INDOUBT command. In addition, manual intervention is necessary to ensure that the LUW is in a consistent state at all sites where the LUW has been distributed. This may require intervention at this database manager, or possibly at another database manager. Manual intervention could mean manually fixing the data or possibly restoring an archive.</p>	<p>Send normal completion reply indicating Heuristic Committed state.</p>

Note:

1. The state **Syncpoint Pending** is not possible at DB2 Server for VM servers. The server completes any sync point actions such as prepare to commit, commit or rollback before the CRR Recovery Server performs any sync point logging.
2. The state **Backout (Reset)** is not possible at DB2 Server for VM servers. The servers complete rollback processing before the CRR Recovery Server performs any sync point logging for backout.
3. The state **committed** is not possible at DB2 Server for VM servers. The servers complete commit processing before the CRR Recovery Server performs any sync point logging for committed.
4. The state **LUWID not found** is possible if (for example) an older DB2 Server for VM archive without an in-doubt LUW is CRR log name, TPN and LUNAME recorded by the restored database matches the current CRR log name, TPN and LUNAME.

Displaying Resynchronization Status using the SHOW CONNECT Command

When DB2 Server for VM is performing *resynchronization initialization* with the CRR recovery server, its status may be seen by the SHOW CONNECT command. The following message is displayed:

```
Recovery Agent is processing Resynchronization Initialization
```

If the database manager is in a communications wait, waiting for the CRR recovery server to reply to its exchange log names request, the following message is displayed:

```
Recovery Agent is processing Resynchronization Initialization
and is in a communications wait with the CRR Recovery Server
```

When the CRR recovery server is performing *resynchronization recovery* with DB2 Server for VM

```
Recovery Agent is processing Resynchronization Recovery
```

If the database manager is in a communications wait, waiting for the CRR recovery server to acknowledge its exchange log names and compare states replies, following message is displayed:

```
Recovery Agent is processing Resynchronization Recovery
and is in a communications wait with the CRR Recovery Server
```

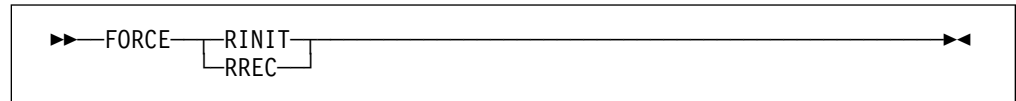
If the database manager is committing or rolling back the logical unit of work requested by the CRR recovery server, the following message is displayed:

```
Recovery Agent is processing Resynchronization Recovery
and is waiting for a <commit|rollback> to complete.
```

This information is available in tokenized format.⁴

Terminating Resynchronization using the FORCE Command

During *resynchronization initialization* or *resynchronization recovery*, the database manager could wait indefinitely for a response from the CRR recovery server. If this is the case, the following command may be used to terminate *resynchronization initialization* processing:



If “FORCE RINIT” is entered, *resynchronization initialization* processing is terminated and the SYNCPTN parameter is changed from Y to N. If “FORCE RREC” is entered, *resynchronization recovery* is terminated and deallocate (abend) is performed to terminate the conversation with the CRR recovery server.

Notes:

1. The operator must issue the SHOW ACTIVE, SHOW CONNECT or SHOW SYSTEM command prior to the FORCE RINIT/RREC command. Otherwise, the following message is issued:

```
ARI0225E System operator must issue SHOW ACTIVE, SHOW CONNECT or
SHOW SYSTEM command prior to FORCE command.
```

and FORCE processing terminates.

2. If the database is not performing *resynchronization initialization* when the FORCE RINIT command is entered, then the following message is displayed:

```
ARI2040E FORCE RINIT may only be entered when Resynchronization Initialization is active.
```

and FORCE processing terminates.

3. If the database is not performing *resynchronization recovery* when the FORCE RREC command is entered, then the following message is displayed:

```
ARI2040E FORCE RREC may only be entered when Resynchronization Recovery is active.
```

and FORCE processing terminates.

4. If the FORCE RINIT command was already issued, the following message is displayed:

```
ARI2041E FORCE RINIT is already scheduled.
```

⁴ For more information about tokenized format, see “Appendix A” of the *Diagnosis Guide and Reference for IBM VM Systemsmanual*.

and FORCE processing terminates. (Note that the scheduled FORCE command remains!)

5. If the FORCE RREC command was already issued, the following message is displayed:

```
ARI2041E FORCE RREC is already scheduled.
```

and FORCE processing terminates. (Note that the scheduled FORCE command remains!)

6. If extra parameters are entered after "FORCE RINIT/RREC," then the following message is displayed:

```
ARI0229E Too many FORCE command input parameters
```

and FORCE processing terminates.

Chapter 16. Using TCP/IP with DB2 Server for VM

Application requesters are able to use TCP/IP to access the DB2 Server for VM application server. DB2 Server for VM application requesters can use private protocol or DRDA remote unit of work protocol over TCP/IP to access the DB2 Server for VM application server. DB2 Server for VM application requesters can use DRDA remote unit of work protocol over TCP/IP to access other DB2 family servers that have TCP/IP support. Other application requesters can use DRDA remote unit of work protocol over TCP/IP to access the DB2 Server for VM application server.

Preparing the Application Server to use TCP/IP

The following must be done to allow the application server to use TCP/IP.

1. TCP/IP for VM must be installed and configured.
2. The TCP/IP client data disk must be accessed. This disk is often defined as TCPMAINT 592. This disk contains the TCPIP DATA file and the ETC SERVICES file which are necessary for successful TCP/IP support initialization.
3. The ARICTCP MODULE must be created and stored on the production disk. The instructions for creating the ARICTCP MODULE are found in the DB2 Server for VM program directory.
4. A C runtime library must be available. The SCEERUN LOADLIB provided with VM/ESA Version 2 Release 2.0 or later is sufficient. The C runtime library provided with LE for VM is also acceptable.
5. Optionally, the ETC SERVICES file must be updated with the RESID of the application server and the port number it should use.

TCP/IP support is invoked at system initialization time. If TCP/IP for VM is available, the server will make use of it. The application server must be able to determine what port number to listen on for connections. This can be accomplished in a number of ways.

1. The ETC SERVICES file on the TCP/IP maintenance disk has a port number associated with the RESID of the application server.
2. The new initialization parameter, TCPPORT, can be used to specify the port number to listen on.
3. The well know port number for ddm-rdb, 446, is used. This name must be defined in the ETC SERVICES file.

Each method has advantages and disadvantages.

The first method of using the ETC SERVICES file is the preferred method. This file is maintained by the TCP/IP administrator and resides on the TCP/IP client data disk. Since many application servers can run on the same VM system, it must be ensured that they do not use the same TCP/IP port since only one server can use a port. Identifying the port numbers in this file makes it easier to ensure that different servers are using different ports. The name used must be the application server's RESID. This can be determined by issuing the SHOW INITPARM command on the application server.

The entry in the ETC SERVICES file consists of a service name, port number, protocol and an optional comment. Valid protocols in the ETC SERVICES file are tcp and udp. The only protocol DB2 Server for VM will recognize is tcp. It cannot understand udp. It must be noted that the searches done on the ETC SERVICES file are case sensitive. When the RESID is used as the search criteria it will be entirely in upper case and must be defined entirely in upper case. Similarly, the protocol is also part of the search criteria and is specified entirely in lower case and must be defined in lower case for the search to be successful.

The following is an example of entries for 2 DB2 Server for VM application servers.

```
SQLPROD          6100/tcp          # Production database machine
SQLTEST          6200/tcp          # Test database machine
```

The second method of port identification is the new initialization parameter, TCPSPORT. The advantage of this is that the ETC SERVICES file does not need to be updated. This is helpful when initially testing TCP/IP support or when TCP/IP support needs to be enabled, but the ETC SERVICES file cannot be updated. The disadvantage is that it is possible that another application may be using the same port.

The third method is the least desirable. If there is no entry for the RESID in the ETC SERVICES file or TCPSPORT was not specified, there is a well known port assignment for relational databases. It is called ddm-rdb and the port number is 446. This name must be defined in the ETC SERVICES file. The file that is shipped with TCP/IP for VM has this definition. This has the advantage of doing no extra configuration to TCP/IP for VM and to the application server. The disadvantage is that only one application server on that VM system can use that definition.

We will take advantage of all three of the methods. The actions can be broken down into three scenarios.

1. If the TCPSPORT parameter is not specified when the application server is started, the application server will search the ETC SERVICES file for a service name whose name matches the application server's RESID. If a port number is found it will use it. If a port number is not found, it will use ddm-rdb to do a look up in the ETC SERVICES file for the well known port number. If a port number is found it will use it. If a port number is not found, then TCP/IP support on the application server will not be used. If an error is returned from any of the TCP/IP functions used to do the look up, it will be assumed that TCP/IP is not available and TCP/IP support on the application server will not be available.
2. If the TCPSPORT parameter is specified when the application server is started, the application server will use this parameter while performing TCP/IP support initialization. No look up in the ETC SERVICES file is done. If the port cannot be used, no attempt is made to find another port. If an error is returned from a TCP/IP function, it will be assumed that TCP/IP is not available and TCP/IP support on the application server will not be available.
3. If the TCPSPORT parameter is specified with a value of 0, TCP/IP support is not initialized at all. If you do not have TCP/IP support on your system or do not want to enable TCP/IP support for an application server, the TCPSPORT=0 parameter should be used.

Preparing the Application Requester to use TCP/IP

The following must be done to allow an application requester to use TCP/IP.

1. TCP/IP for VM must be installed and configured.
2. The TCP/IP client data disk must be accessed. This disk is often defined as TCPMAINT 592. This disk contains the TCPIP DATA file and the ETC SERVICES file which are necessary for successful TCP/IP support initialization.
3. The ARICTCP MODULE must be created and stored on the production disk. The instructions for creating the ARICTCP MODULE are found in the DB2 Server for VM program directory.
4. A C runtime library must be available. The SCEERUN LOADLIB provided with VM/ESA Version 2 Release 2.0 or later is sufficient. The C runtime library provided with LE for VM is also acceptable.
5. Optionally, the ETC SERVICES file must be updated with the RESID of the application server and the port number it should use.
6. The CMS communications directory, COMDIR, must be updated for TCP/IP.
7. The CEEPIPI MODULE may have to be rebuilt, to prevent the user exit CBEXIT from clearing the SYSIN filedef when the LE environment is terminated. If this is not done, file not open or file not found errors will be received from programs that access a database using TCP/IP and have defined a SYSIN filedef for input. SQLPREP and SQLDBSU are two such programs. Instructions for modifying the CEEPIPI MODULE are found in the DB2 Server for VM program directory.

To indicate that TCP/IP is to be used to establish a connection from the application requester, the communications directory, COMDIR, is used. If the COMDIR entry for the database that is the target of the SQL CONNECT statement has **host** and **service** entries defined, then TCP/IP will be used to establish the connection. If a DB2 Server for VM application requester wants to access a remote database using TCP/IP, an entry in the communications directory, COMDIR, must be set up.

The COMDIR must be set up to provide the host and service names the requester will use on the connection. A TCP/IP COMDIR entry will look like the following:

```
:nick.TCPVM1  :service.SQLTEST
               :host.TORVMLB6
               :security.PGM
               :userid.USERID
               :password.PASSWORD
               :dbname.SQLTEST
:nick.TCPVM2  :service.6100
               :host.9.21.31.109
               :security.PGM
               :userid.USERID
               :password.PASSWORD
               :dbname.SQLPROD
```

The tpn tag is replaced with the service tag and the luname tag is replaced with the host tag. The modename tag is not needed because it is an SNA network parameter. An IP address can be specified instead of a host name on the host tag and a port number can be specified on the service tag instead of a service name. This will bypass host name and service name lookup calls.

When the COMDIR search is done on the dbname, if the service and host tags are present, then TCP/IP will be used. If the tpn and lname tags are present, then SNA will be used. If both are present an error, SQLCODE = -841, SQLERRM = X'0015', will be issued. If the modename tag appears with the host and service tags, it will be ignored.

The security, userid and password tags will be used for both SNA and TCP/IP protocols. If TCP/IP communications is being used, the user cannot use APPCPASS directory statements for specifying the userid and password. APPCPASS is only used by SNA LU 6.2 communications.

The nick tag is still not used. The search of the COMDIR is based on the dbname tag and the first matching entry is used.

Since the COMDIR search is not performed until the user issues an SQL CONNECT statement, either implicitly or explicitly, any errors related to COMDIR setup problems must be reflected in an SQLCA returned to the application requester after the CONNECT statement. This will be reflected as SQLCODE = -841, A communications directory error has occurred. Reason Code=X'n'. The reason code will indicate the nature of the error.

Security Considerations for TCP/IP

Conversation security is part of SNA communications and prevents a physical connection if the security check fails. TCP/IP communications has no security checking. It is up to the server to determine if it will accept or reject a connection from a client. The physical connection must initially be accepted and security information must be transmitted and checked. If the security check fails, the server must close the connection.

If DRDA protocol is being used, the ACCSEC and SECCHK DDM data streams have been added to the DRDA architecture to handle this. They determine the security mechanism that will be used to validate the user and they provide the userid and password information. The security mechanism, userid and password are taken from the COMDIR entry.

If SQLDS protocol is being used, an explicit connect request is sent to the application server. The userid and password are taken from the COMDIR entry.

The risk with TCP/IP communications is that it allows an unencrypted password to be transmitted over the network. Two options exist to avoid this problem. The first is to allow a user be already verified. The second is to use an external security manager that will provide an encrypted password that can be safely transmitted on the network.

If a user is already verified, only a user id must be transmitted. The target application server must be configured to accept already verified users. DB2 Server for VM provides the SECALVER initialization parameter. If SECALVER=Y, the DB2 Server for VM applications server will accept a TCP/IP connected user and only validate the userid. If SECALVER=N and a connection request comes in with only a userid, the connection will be rejected with a missing password error.

A DB2 Server for VM application requester would configure the security tag in the COMDIR to SAME and only specify a userid tag. The password tag would be

omitted. If a password is included, it is used and verified even if security SAME was specified.

If the application requester can generate an encrypted password, the target application server must be able to verify it. DB2 for OS/390 can use RACF PassTickets for this purpose. If a DB2 for OS/390 application requester wants to connect to a DB2 Server for VM application server using TCP/IP and is using PassTickets, the DB2 Server for VM application server must be able to verify the userid and the PassTicket. The DB2 Server for VM initialization parameter, SECTYPE=ESM, is used for this purpose. Specifying this parameter will establish a connection to an external security manager. This external security manager must support the RACROUTE interface. RACROUTE and PassTickets are supported by RACF V1R10 for VM (5740-XXH) or later. The DB2 Server for VM application server would pass the userid and PassTicket to the ESM to be verified. The ESM could also be used to verify users and unencrypted passwords, but its real value is in the use of PassTickets.

The default value for SECTYPE is DB2. If this is used, then userid and password verification is done by checking the values in the SYSUSERAUTH catalog table.

The SECALVER and SECTYPE parameters are only used if the incoming connect request is via TCP/IP from any type of requester or if the incoming connect request is from a DB2 Server for VSE DRDA application requester via SNA. DB2 Server for VSE does not support any connections via TCP/IP.

If only a userid is sent on the connect request, the DB2 Server for VM application server checks the value of the SECALVER parameter. If the value is N, the connection is rejected. SECALVER = N means the user is not already verified. A userid and password are required for verification. If the value is Y and only a userid is sent, the connection is accepted. SECALVER = Y means the user is already verified. Only a userid is required for verification. In this case the userid is verified based on the SECTYPE value. If SECTYPE = DB2, the SYSTEM.SYSUSERAUTH table is checked if the user has connect authority. The password column is not checked. If SECTYPE = ESM, the userid is passed to the ESM via RACROUTE. The password is not checked.

If a userid and password are sent on the connect request, the DB2 Server for VM server ignores the value of SECALVER and only checks the value of SECTYPE. The userid and password are both required for verification. The verification is based on the SECTYPE value. If SECTYPE = DB2, the SYSTEM.SYSUSERAUTH table is checked if the user with that password has connect authority. If SECTYPE = ESM, the userid and password is passed to the ESM via RACROUTE. The userid and password are both checked. In this case the password can be a PassTicket. The way the ESM was set up determines if it is expecting a password or PassTicket from this userid.

If SECTYPE = ESM is specified, a check is performed during initialization to ensure that the external security manager is available and that it supports the RACROUTE interface. On VM, the command **RPIUCMS INIT** is issued. The expected response is **RPICMS016I USER/RACF VM communication path established**. If this response is not received, an error message, ARI4108E Unable to initialize the external security manager is returned. The value of SECTYPE is switched to DB2 and processing continues.

Application Requester

The application requester must provide a userid and password to successfully connect to an application server. On VM, applications can specify the userid and password in the following ways:

1. The "CONNECT userid IDENTIFIED BY password" statement
2. Utilizing a CMS Communications directory (COMDIR)
3. Using APPCPASS CP directory statements in conjunction with a COMDIR

In private flows, all methods may be used. In DRDA flows, only the last two methods can be used. In DRDA flows over TCP/IP, only method 2 is used.

In the VM application requester if the connection to be established is DRDA via TCP/IP, security handshaking datastreams are sent. If the connection is to be established via SNA or private flow via TCP/IP, no security handshaking datastreams are sent. This is determined by the values in the COMDIR and the value of the PROTOCOL parameter of SQLINIT.

SQLINIT PROTOCOL	Connection Protocol	Security Handshaking Done?
SQLDS	SNA	No
SQLDS	TCP/IP	No
AUTO	SNA	No
AUTO	TCP/IP	Yes
DRDA	SNA	No
DRDA	TCP/IP	Yes

SECTYPE = ESM allows the user to use an external security manager to authenticate TCP/IP connections to the database manager. If the external security manager supports PassTickets, they can be used instead of a password to avoid having an unencrypted password sent on a TCP/IP network.

If SECTYPE = ESM is going to be used, two major set up steps of the external security manager must be completed.

1. Enable the application server to access the external security manager
2. Add the user connect authorizations that were formerly in the SYSTEM.SYSUSERAUTH catalog table to the external security manager.

If SECTYPE = ESM is going to be used, the external security manager must be set up to allow the application server to access it. Refer to the *External Security Interface (RACROUTE) Macro Reference for MVS and VM* manual for details. See the section titled "Authorization to Issue RACROUTE Requests".

The instructions given here only describe the minimum steps required. Refer to the above manual for any detailed instructions.

1. Identify the RACF service machine to which RACROUTE requests will be sent. This is done via the RACF SERVMACH file which is typically installed on the CMS Y-disk.
2. In this case, the RACROUTE issuer is the VM machine for the DB2 Server for VM server. Update the RACROUTE issuer's CP directory entry by adding an IUCV card which specifies the RACF service machine with which the RACROUTE issuer will be communicating. For example,


```
IUCV RACFVM PRIORITY MSGLIMIT 255
```
3. RACF-authorize a connection to the RACF service machine
 - Log on with a user ID having the system-SPECIAL attribute
 - Create a profile named ICHCONN in the FACILITY class:


```
RDEFINE FACILITY ICHCONN UACC(NONE)
```
 - Give READ or UPDATE access authority to appropriate service machines:


```
PERMIT ICHCONN CLASS(FACILITY) ID(userid | groupid)
ACCESS(appropriate-access)
```

Appropriate-access should be READ or UPDATE. If NONE is specified, use of the RACROUTE macro interface is not allowed.
 - Activate the FACILITY class (if this class is not already active):


```
SETROPTS CLASSACT(FACILITY)
```
4. Follow the procedures described in "Issuing RACROUTE Requests on CMS" or "Issuing RACROUTE Requests on GCS" to set up the environment to issue RACROUTE requests on CMS or GCS, respectively.

The RPIUCMS module referred to in the procedures is available in the RACF product, not the VM operating system. If you install another external security product on VM, that external security product should provide an equivalent RPIUCMS function as described in Appendix C, "RACROUTE Interface to an External-Security-Manager Product (Non-RACF) on VM." in the *External Security Interface (RACROUTE) Macro Reference for MVS and VM* manual.

If PassTickets will be used instead of passwords, a Secured Signon application profile must be defined for the application in the PTKTDATA class. In addition, RACF must be at Version 1 Release 10 or later.

The user must generate a PassTicket before issuing the SQL CONNECT statement. The PassTicket is then used in the SQL CONNECT statement instead of the password. See the *RACF Macros and Interfaces* manual for details on how a PassTicket is generated.

If SECTYPE = DB2 is to be used, the users must be defined in the SYSTEM.SYSUSERAUTH catalog. This is done by a user with DBA authority using the SQL GRANT CONNECT command. See the *DB2 Server for VSE & VM SQL Reference* manual for details.

Appendix A. Virtual and Real Storage Requirements

This appendix offers guidelines for estimating the resources needed for the database manager.

For a system operating in multiple-user mode and processing both preplanned and dynamic requests, adequate performance requires:

- Approximately 3MB to 4MB of real storage for a database virtual machine and its buffers. Sufficient real storage must be made available such that there is as little paging of the database machine virtual storage as possible. The corresponding virtual storage is 8MB for the base code, or the base code and DSS code; 10MB for DRDA code.
- Approximately 30KB to 100KB of real storage for a user machine running DB2 Server for VM preplanned applications, the preprocessors, the DBS utility, or ISQL. The corresponding virtual storage requirement is about 6 megabytes.

Initial Virtual Storage Requirements of Components

The virtual storage requirements for the database manager are its sum of the static and dynamic storage requirements. Static storage is used for the database manager code, database I/O control blocks, the page and directory buffers, and other reasons. Dynamic storage is used for the preprocessing and execution of SQL statements.

Table 36 on page 444 and Table 37 on page 445 show the static storage required in a database machine to start the application server.

The storage required for these purposes can vary greatly, depending on the complexity of the SQL statements that, the size of the package, and the number of packages in virtual storage.

The minimum dynamic storage required is $NCUSERS \times 130000$. If the DRDA code is installed, it is $NCUSERS \times 180000$.

Table 36. Initial Storage Requirements of DB2 Server for VM Database Machines

Formula/Constraints	Recommended Minimums for	
	Single-User Mode	Multiple-User Mode
DB2 Server for VM Code ^{1, 14}	3324000	3077000
Message Repository ¹⁴	250000	250000
DB2 Server for VM Control Blocks		
• Base	140000	140000
• Lock Request Blocks NLRBS x 24 ^{2, 3, 5}	12288	60480
• Concurrent Scans Storage NCSCANS x 50 ^{2, 3} x (NCUSERS+3) ^{2, 6}	4500	12000
• Miscellaneous LRBs 8 x (First Prime Number > (NLRBS/4))	1048	5048
• Static Agent Storage below 16M NCUSERS x 57500 ¹⁴	57500	287500
• Static Agent Storage anywhere NCUSERS x 50000	50000	250000
• Working Storage Area ¹⁴	109000	109000
• Working Storage Queue Headers ^{2, 3} 32 + (2 + 2 x (NCUSERS + 3) + NPACKAGE x (NCUSERS + 1)) x 68	848	5336
• Package Control Blocks NCUSERS x NPACKAGE x 72 bytes ^{2, 3}	72	3600
• Communications Control Blocks MAXCONN x 336 ^{7, 13}	21168	21168
• If TCP/IP Communications is used, add: (NCUSERS+2) x 264 ¹⁶	0	0
• Accounting Control Blocks (MAXCONN/51+1) x 4096 ⁷	9156	9156
• Stored Procedure Caches (76 * the number of stored procedure servers) +(32 * the number of stored procedure server groups) +(for each procedure, 116 +the length of the LE options string (max 254) + (68 * the number of parameters))	0	0
DB2 Server for VM Buffers		
• 4 x Max of ((the 1st power of 2 that is >= (NPAGBUF/8)) or 256) + NPAGBUF x 4144 ^{2, 3, 10}	59040	125344
• 4 x Max of ((the 1st power of 2 that is >= (NDIRBUF/8)) or 256) + NDIRBUF x 560 ^{2, 3, 11}	8864	17824
Database Control Blocks		
• Base control blocks	88000	88000
• MAXDBSPC x 8 ¹²	8000	8000
• MAXPOOLS x 18 ¹²	576	576
• MAXEXTNT x 6 ¹²	384	384
Database I/O Control Blocks		
• NCUSERS x 216 x # of minidisks ⁴	2808	14040
• Minimum of (64 or NPAGBUF/10) x (NCUSERS+3) x 16 ¹⁵	336	384
	----- 4147588 ^{8, 9}	----- 4484840 ^{8, 9}

Note: The storage requirements above do not include the virtual storage used by the user program or CMS. (Even if CMS is defined as a saved system, CMS still uses some storage in the user's virtual machine.)

The Stored Procedure example assumes that no stored procedures or stored procedure servers have been defined.

Note:

The above numbers are in bytes.

¹ If the DRDA code is installed on the application server, an additional 1.8 megabytes is required (this applies to both multiple-user mode and single-user mode); if it is installed on the application requester, an additional 1.8 megabytes is required for single-user mode.

² The numbers for multiple-user mode assume the following defaults: NCUSERS (5), NPACKAGE (10), NCSCANS (30), NLRBS (2520), NLRBU (1000), NPAGBUF(30), NDIRBUF(30), PROTOCOL=SQLDS and TCPPOINT not equal to zero.

³ The numbers for single-user mode assume the following defaults: NCUSERS (1), NPACKAGE (1), NCSCANS (30), NLRBS (512), NLRBU (1000), NPAGBUF(14), NDIRBUF(14) and TCPPOINT is not applicable.

⁴ The number of minidisks = 13 (10 dbextents + 1 directory minidisk + 2 logs for dual logging)

⁵ The calculation for NLRBS is:
 $NCUSERS \times 2 + (NCUSERS \times NLRBU)/2 + 10$.

⁶ In single-user mode, $NCSCANS \times 50 \times 3$.

⁷ Assume VM MAXCONN is 63 (13 for minidisks and 50 for user connections).

⁸ If you have coded an accounting exit, add its size to the total virtual storage requirements for both single and multiple-user mode. If you have coded any field procedures, add their total size to the requirements for both single and multiple-user mode. In addition, add NCUSERS times the maximum working storage used by any of your field procedures. For more information, see "Field Procedures" on page 387.

⁹ Add the dynamic storage requirement of active agents.

¹⁰ The calculation for the default NPAGBUF is: $10 + 4 \times NCUSERS$

¹¹ The calculation for the default NDIRBUF is: $10 + 4 \times NCUSERS$

¹² The recommended minimum assumes the defaults for MAXDBSPC (1000), MAXPOOLS (32), and MAXEXTNT (64).

¹³ If using PROTOCOL=AUTO, the Communications Control Blocks calculation is: $MAXCONN \times 552$.

¹⁴ The following always resides below 16 megabytes: 1.2 megabytes of the DB2 Server for VM Code, the Message Repository, the Static Agent Storage below 16 megabytes, and the Working Storage Area.

¹⁵ In single-user mode, minimum of $(256 \text{ or } NPAGBUF/2) \times 48$.

¹⁶ TCP/IP communications is only used in multiple-user mode, so the single-user mode value is always zero.

User Machine Components

Table 37 shows the virtual storage requirements in a user machine using the base code only. A 6MB machine size is recommended.

Table 37 (Page 1 of 2). Virtual Storage Requirements of Components in a User Machine

Resource adapter	242 (Used for any DB2 Server for VM facility or user program)
------------------	---

Table 37 (Page 2 of 2). Virtual Storage Requirements of Components in a User Machine

ISQL	562 (320 for ISQL + 242 for the resource adapter)
DBS utility	462 (220 for the DBS utility + 242 for the resource adapter)
Any DB2 Server for VM Preprocessor	464 (222 for the preprocessor + 242 for the resource adapter)
Note: The storage requirements above do not include the virtual storage used by the user program or CMS. (Even if CMS is defined as a saved system, CMS still uses some storage in the user's virtual machine.)	

Notes:

1. All numbers are in kilobytes.
2. The resource adapter uses a minimum of 242 kilobytes of virtual storage. If the DRDA code is installed in the user machine, an **extra** 1726 kilobytes are required.

These figures assume that national language message repositories are in saved segments. For each one not in a saved segment, add 250 kilobytes. In addition, if you use a VM/ESA operating system, add 5 kilobytes for the second and each subsequent CMS work unit used by the application.

You should add this value to the virtual storage requirements of the SQL application to arrive at the size of the user machine.
3. The DBS utility requires 462 kilobytes.
4. To operate any DB2 Server for VM preprocessor in a user machine, you would need at least 464 kilobytes of free storage. To operate a preprocessor on the database machine in single-user mode you would add 222 kilobytes to the size of the machine.
5. If CMS is not defined as a saved system, its size must be added when calculating the storage requirements.
6. ISQL cannot be run in the same machine as the database manager.
7. For user programs, if the fetch/insert blocking option is in effect, add 8 kilobytes for each concurrently open cursor that qualifies for blocking.
8. If you have coded an accounting exit, a cancel exit, or a TRANSPROC, add its size to the total virtual storage requirements.

The amount of virtual storage required in the user machine for ISQL varies. The following variables should also be considered:

- Use of SELECT
- Use of routines or EXECs.

Use of SELECT

The row length is the only variable that affects the amount of storage needed by a SELECT result. The number of rows retrieved from a SELECT has no effect on the amount of storage required.

ISQL gets a buffer for a certain number of rows before retrieving any rows from tables. The number of rows depends on the size of the terminal screen. Specifically, ISQL gets a buffer to hold (SCREEN_HEIGHT - 4)*2 row -- that is, enough storage for two screens of rows. (Four is subtracted from the screen height because of the column header and input/status areas of the screen.) If more than two screens of rows are retrieved, this buffer is reused.

The number of columns referenced in a SELECT command in ISQL is limited to 45. ISQL uses the length of the retrieved columns to determine the buffer size not the number of column references. The storage needed is the number of rows for two screens multiplied by the length of one row. (Note that the length of the row may be larger than the length of the screen.) If the storage required for the buffer exceeds 32 kilobytes, not all selected columns are displayed, or the user receives an error message.

The SET VARCHAR and FORMAT VARCHAR ISQL commands will change the size of the row and thus the size of the buffer.

Use of Routines or EXECs

If you use ISQL routines or stack ISQL commands in an EXEC, the amount of virtual storage used in the user machine increases, because VM program stacks are used. Thus, the amount of storage required is related to the number of lines in the routines (or EXECs). For more information on program stacks, see the *VM/ESA REXX/VM User's Guide* manual.

Appendix B. Estimating Database Storage

This appendix describes procedures for estimating the size of the directory, the SYS0001 dbspace and ISQL dbspace.

For information on estimating the size of user dbspaces, see the *DB2 Server for VM Database Administration* manual.

Storage Capacities of IBM DASD Devices

The effective storage capacities of IBM DASD devices vary, depending on how the devices are being used. The database manager uses VM DASD block I/O services for managing DASD space for the directory minidisk, the log (or logs), and the storage pool minidisks (dbextents). The log and dbextent minidisks are managed with 4 kilobyte blocks. The directory minidisk uses 512-byte blocks.

The following figure indicates how many 512 and 4 096 byte blocks can be allocated on a CMS minidisk. The first number is the number of usable blocks, and the number in parentheses is the total number of blocks. The difference between the two numbers is the blocks that CMS allocates for its file management.

Device	Cylinders or Blocks for Each Volume	4 096-byte Blocks for Each Cylinder	4 096-byte Blocks for Each Volume	Ratio of 512-byte to 4096-byte	512-byte Blocks for Each Volume
3375	959	96	91,960 (92,064)	40/8	456,462 (460,320)
3380 J	885	150	132,602 (132,750)	46/10	605,531 (610,650)
3380 E	1,770	150	265,214 (265,500)	46/10	1,211,076 (1,221,300)
3380 K	2,655	150	397,827 (398,250)	46/10	1,816,621 (1,831,950)
3390-1	1,113	180	200,122 (200,340)	735/180	811,205 (818,055)
3390-2	2,226	180	400,254 (400,680)	735/180	1,622,418 (1,636,110)
3390-3	3,339	180	600,387 (601,020)	735/180	2,433,631 (2,454,165)
3390-9	10,017	180	1,801,177 (1,803,060)	735/180	7,300,915 (7,362,495)
3370-1	558,000	N/A	69,667 (69,750)	8/1	553,321 (558,000)
3370-2	712,752	N/A	88,993 (89,094)	8/1	706,780 (712,752)
9332-400	360,036	N/A	44,948 (45,004)	8/1	357,018 (360,036)
9332-600	554,800	N/A	69,268 (69,350)	8/1	550,149 (554,800)
9335	804,714	N/A	100,433 (100,588)	8/1	797,974 (804,714)
9336-010	920,115	N/A	114,885 (115,014)	8/1	912,411 (920,115)
9336-020	1,672,881	N/A	208,883 (209,110)	8/1	1,658,881 (1,672,881)
9336-025	1,672,881	N/A	208,883 (209,110)	8/1	1,658,881 (1,672,881)
9345-1	1,440	150	215,767 (216,000)	123/30	878,183 (885,600)
9345-2	2,156	150	323,056 (323,400)	123/30	1,314,843 (1,325,940)
0671	574,560	N/A	71,735 (71,820)	8/1	569,743 (574,560)

Table 39 and Table 40 show the capacities of IBM devices for storing log and dbspace pages (dbextent space). Table 41 and Table 42 show the capacities for storing directory information.

Table 39. Log and Dbextent Storage Capacities of IBM Count-Key-Data DASDs

DASD Type	Number of Cylinders	Tracks for Each Cylinder	Megabytes for Each Cylinder	Megabytes for Each Volume
3375	959	12	0.3749	359
3380 J	885	15	0.5858	518
3380 E	1,770	15	0.5858	1,036
3380 K	2,655	15	0.5858	1,555
3390-1	1,113	15	0.7031	782
3390-2	2,226	15	0.7031	1,565
3390-3	3,339	15	0.7031	2,347
3390-9	10,017	15	0.7031	7,041
9345-1	1,440	15	0.5858	843
9345-2	2,156	15	0.5858	1,262

Table 40. Log and Dbextent Storage Capacities of IBM FB-512 DASDs

DASD Type	Megabytes for Each Volume	4 Kilobyte Pages for Each Volume
3370-1	272.4	69,750
3370-2	348.0	89,094
9332-400	175.7	45,004
9332-600	270.8	69,350
9335	392.9	100,589
9336-010	449.2	115,014
9336-020	816.8	209,110
9336-025	816.8	209,110
0671	280.5	71,820

Table 41. Directory Storage Capacities of IBM Count-Key-Data DASDs

DASD Type	Number of Cylinders per Volume	Tracks for Each Cylinder	Megabytes for Each Cylinder	Megabytes for Each Volume
3375	959	12	0.2343	224
3380 J	885	15	0.3295	291
3380 E	1,770	15	0.3295	583
3380 K	2,655	15	0.3295	874
3390-1	1,113	15	0.3645	405
3390-2	2,226	15	0.3645	811
3390-3	3,339	15	0.3645	1,217
3390-9	10,017	15	0.3645	3,651
9345-1	1,140	15	0.3002	432.28
9345-2	2,156	15	0.3002	647.23

DASD Type	Megabytes for Each Volume	512-Byte Blocks for Each Volume
3370-1	272.4	558,000
3370-2	348.8	712,752
9332-400	175.7	360,036
9332-600	270.8	554,800
9335	392.9	804,714
9336-010	449.1	920,115
9336-020	816.8	1,672,881
9336-025	816.8	1,672,881
0671	280.5	574,560

These capacity charts are referenced in later calculations for determining data set allocations of the directory, log, and dbextent minidisks.

Table 43 shows the minimum space allocations for a log or dbextent minidisk.

DASD Type	Minimum Space Allocation
3375	1 cylinder
3380	1 cylinder
3390	1 cylinder
9345	1 cylinder
FBA	528 Blocks

Determining Equivalent Minidisk Sizes on Different Device Types

This section shows an example of how to determine approximately equal minidisk sizes on different devices. The example uses the numbers shown in Table 38 on page 450. Depending on the block size of the minidisk you are moving to, use the 512-byte column, or the 4096-byte column.

The example assumes these things:

- You have 7 cylinders on a 3375 DASD
- You are moving to a 3380 DASD
- You are moving your directory minidisk.

The directory minidisk uses 512-byte blocks.

To determine how many cylinders you need on the new device, you must first find out how many blocks you have for each volume on your current device and the new device. Then, you use these numbers to find out how many cylinders you will need on the new device. To do this, you solve the equation in “Determining the Number of Blocks or Cylinders on Your Current Device” on page 453.

If you are moving from an FB-512 device to a count-key-data device, you already know how many blocks you have on your current device. You only have to solve

the new device side of the equation in “Determining the Number of Blocks or Cylinders on Your Current Device” on page 453 to find out how many cylinders you need on the new device. (If you use 4 096-byte blocks, divide the number of blocks on your FB-512 device by 8 before you do the equation.)

If you are moving from a count-key-data device to an FB-512 device, you only need to know how many blocks you will need on the new device. You only have to solve the current device side of the equation in “Determining the Number of Blocks or Cylinders on Your Current Device” to find out how many blocks you need on the new device. (If you use 4 096-byte blocks, multiply the number you solve for in the equation by 8 to find out how many blocks you need on the FB-512.)

Determining the Number of Blocks or Cylinders on Your Current Device

The example uses this equation:

$$\frac{a * b}{c} = \frac{x * d}{e}$$

In the equation, you solve for x to find the number of blocks or cylinders required on your new device. The other letters are variables, which are determined by the values in Table 38 on page 450. The letters stand for the following attributes:

- a** The number of cylinders on your current device
- b** The number of blocks for each volume on your current device.
- c** The number of cylinders for each volume on your current device.
- d** The number of blocks for each volume on your new device.
- e** The number of cylinders for each volume on your new device.
- x** The unknown in this equation, which stands for the number of blocks or cylinders on your new device.

With 7 cylinders on a 3375, and a block size of 512 bytes to be moved to 3380, you would use the 512-byte column of Table 38, giving:

$$\frac{7 * 456462}{959} = \frac{x * 605531}{885}$$

By following these steps, you can solve for x :

1.
$$x = \frac{7 * 456462 * 885}{959 * 605531}$$
2. $x = \text{approx. } 4.87$

To ensure that you allow enough space on the new device, increase x by 5% and round up to the next whole number. In this example, you would need 6 cylinders on the new device type.

Relationship of Megabytes to 4-Kilobyte Pages

In the database generation process, all dbspace and dbextent DASD space definitions are expressed in terms of 4-kilobyte pages: that is, each page represents 4096 bytes of storage space. Storage space is used not only for data, but also for indexes and free space initially reserved to facilitate the insertion of new data after the database is in operation.

Space needs are often expressed in terms of megabytes (1,048,576 bytes). Table 44 shows the number of 4-kilobyte pages needed to support a range of megabytes. The dbspace definitions are made in multiples of 128 pages. An alternative to using Table 44 is to use the formula:

Number of 4-kilobyte pages = 256 x number of megabytes

Table 44. Megabytes of Data on 4-Kilobyte Pages

Megabytes	4-Kilobyte Pages
0.0 - 0.5	128
0.5 - 1.0	256
1.0 - 1.5	384
1.5 - 2.0	512
2.0 - 2.5	640
2.5 - 3.0	768
3.0 - 3.5	896
3.5 - 4.0	1,024
4.0 - 4.5	1,152
4.5 - 5.0	1,280
5.0 - 5.5	1,408
5.5 - 6.0	1,536
6.0 - 6.5	1,664
6.5 - 7.0	1,792
7.0 - 7.5	1,920
7.5 - 8.0	2,048
8.0 - 8.5	2,176
8.5 - 9.0	2,304
9.0 - 9.5	2,432
9.5 - 10.0	2,560
50.0	12,800
100.0	25,600
500.0	128,000

Estimating Directory Space Requirements

The required size of the database directory depends on the maximums you established on the MAXPOOLS, MAXEXTNT, and MAXDBSPC parameters during database generation. The directory must be large enough to hold page table entries for the maximum size of the database. Figure 123 shows a formula for calculating the recommended size of a directory minidisk.

Directory size = 7 558 + 16	x MAXDBSPC value
(in bytes)	+ 16 x MAXEXTNT value
	+ 4 x MAXPOOLS value
	+ 0.0021 x Maximum database size

Figure 123. Formula for Calculating Directory Size (in Bytes)

To estimate the value for the maximum database size, determine how many dbspaces (public, private, and internal) your database will need, and the number of pages needed by each dspace; then multiply the total number of pages by 4096 to get the number of bytes. (You may want to overestimate this value to allow for creating unplanned dbspaces, and for increasing the number and size of internal dbspaces.) Finally, multiply this number by 0.0021, to determine how many bytes are needed in the directory to support these dbspaces. The result of this calculation includes the space needed for shadow paging.

Once you have the directory size, you can use the charts shown in the section “Storage Capacities of IBM DASD Devices” on page 449 to determine the minidisk size specifications, in cylinders or blocks, of the device to be used.

Note: Although you do not have to specify the maximum database size during database generation, the size specified for the directory minidisk effectively establishes the limit.

Estimating Storage Pool Requirements

For estimating storage pool sizes, you need to estimate:

- The size of used portions of dbspaces. This includes tables, indexes, and free space on used dspace pages.
- Shadow paging requirements. This is an estimate of the number of dspace pages that can change between checkpoints.

To estimate the number of pages required for a storage pool use the following formula:

$$\begin{aligned} \text{Pool pages} &= 8 \times \text{Number of dbspaces} \\ &+ 1.5 \times \text{Data pages for all dbspaces in the pool} \\ &+ \text{Data pages for the largest table in the pool} \end{aligned}$$

This calculation covers header pages and pages required for table rows and indexes on those tables. If you have increased your dspace data pages value to accommodate future growth of tables, you can decrease the pool pages correspondingly.

The addition of the factor of data pages for the largest table in the pool should accommodate storage pool demands for shadow paging. This allows for UNLOAD and RELOAD of the largest table in the storage pool.

Estimating SYS0001 Dbspace Requirements

The PUBLIC.SYS0001 dbspace is reserved for the catalog tables during database generation, and cannot be redefined. You establish its size (and storage pool) when you generate the database. The size should be large enough to hold all of your database catalog information for the life of the database.

Note: Physical space is not actually consumed until it is required. Consequently, you can define the SYS0001 dbspace to be very large without penalty. Be generous. The penalty for defining the SYS0001 dbspace too small is that, when it has no more usable space, you must completely regenerate the database. This can be a considerable task for a production database. For more information, see “Preparing for Database Regeneration” on page 29.

The formula shown in Figure 2 on page 22 should provide ample storage space for most uses of the database manager. The formula was derived based on a set of assumptions that may not be valid for your database. Review the assumptions and modify the general formula if the assumptions do not accurately represent your planned usage of the database manager.

The following sections describe:

- SYS0001 storage estimating general formula assumptions

You should review these assumptions to determine whether they apply for your planned usage of the database manager. If they do not, you should modify the assumptions (and the resulting formula) to more accurately represent your planned usage.

- Derivation of the general formula for SYS0001 storage estimating
- Formula for SYS0001 storage estimating

This formula is described in “Formula for SYS0001 Storage Estimating” on page 461.

- Examples of using the SYS0001 storage estimating formula

These examples show how to use the SYS0001 storage estimating formula based on three example situations.

- Modifying the SYS0001 storage estimating general formula

This section provides the formulas used to derive the general formula. You can modify the general formula if you want to change some of the assumptions made in deriving the general formula.

SYS0001 Storage Estimating General Formula Assumptions

The general formula for SYS0001 storage estimating was derived based on:

- Average row lengths for catalog rows
- The number of rows required for each object type in the formula.

Average Row Lengths for Catalog Table Rows

Table 45 shows the length of the fixed portions of catalog rows, the maximum stored row length for each catalog table, and an average row length for each of the catalog tables. The average row length is the length assumed in developing the general formula for estimating catalog storage space requirements.

Catalog Table	Minimum Length	Maximum Length	Estimated Average Length
SYSACCESS	46	90	52
SYSCATALOG	64	385	170
SYSCCSIDS	39	39	39
SYSCHARSETS	393	411	400
SYSCOLAUTH	46	82	72
SYSCOLSTATS	27	123	59
SYSCOLUMNS	54	398	156
SYSDBSPACES	40	58	46
SYSDROP	13	13	13
SYSINDEXES	62	232	131
SYSKEYCOLS	55	91	67
SYSKEYS	77	113	89
SYSOPTIONS	11	301	100
SYSPARMS	82	82	82
SYSPROGAUTH	46	54	49
SYSPSERVERS	11	281	60
SYSROUTINES	58	581	170
SYSSTRINGS	286	286	286
SYSSYNONYMS	26	62	36
SYSTABAUTH	57	101	84
SYSUSAGE	36	72	51
SYSUSERAUTH	35	35	35
SYSVIEWS	20	293	200

In Table 45, the minimum and maximum row lengths for each catalog table are determined using the description of the catalog tables in the *DB2 Server for VM Database Administration* manual. The length of a row depends on the data type of each column in the catalog table. The minimum length for each column is found using these values for each data type.

<i>Table 46. Minimum Column Length</i>	
Data Type	Value
DBAINT	4
DBAHW	2
INTEGER	4
SMALLINT	2
CHAR(<i>n</i>)	<i>n</i>
TIMESTAMP	10
VARCHAR(<i>n</i>)	1

Note: The data types DBAINT and DBAHW are used internally by the database manager. Externally, they look like the data types INTEGER and SMALLINT.

For CHAR columns, the length is the column length (*n*). The column lengths are added. For each column that can contain nulls, 1 is added to this figure. The value 8 is then added to this total for catalog table overhead. The resulting number is the minimum row length for the catalog table.

The maximum length for each column is found using these values:

<i>Table 47. Maximum Column Length</i>	
Data Type	Value
DBAINT	4
DBAHW	2
INTEGER	4
SMALLINT	2
CHAR(<i>n</i>)	<i>n</i>
TIMESTAMP	10
VARCHAR(<i>n</i>)	<i>n</i> + 1

For CHAR columns, the length is the column length (*n*). For VARCHAR columns, the length is the maximum column length plus one (*n* + 1). For each column that can contain nulls, 1 is added to this figure. The value 8 is then added to this total for catalog table overhead. The resulting number is the maximum row length for the catalog table.

The average length for each column is calculated this way for most catalog tables:

$$\frac{(\text{maximum length} - \text{minimum length})}{3} + \text{minimum length}$$

This produces a number one third of the way between the minimum and maximum lengths. In some situations, higher values are used because those columns are typically longer. An example is the SYSTEM.SYSVIEWS catalog table, where the VIEWTEXT column contains the command used to create the view. Because these commands are usually over 100 bytes long, a number one third of the way between

the minimum and maximum lengths of the column would be too low. In this situation, the number 200 is chosen arbitrarily.

If you make your own estimates of catalog table row lengths (using the chart provided in Table 52 on page 464), you should choose values for the average row lengths that are accurate for your database. Otherwise, you could underestimate the size of the SYS0001 dbspace. In particular, you should not underestimate the average length of rows in the SYSTEM.SYSCOLUMNS catalog table. If you use the REMARKS or CLABEL columns of this catalog table, your average row length could be far greater than the number (156) given in Table 45 on page 457. Because the SYSTEM.SYSCOLUMNS table can become quite large (it has a row for every column in every table in the database), its size is a major factor in the size of the SYS0001 dbspace.

Assumptions on the Number of Catalog Table Rows

The average number of rows for each catalog table was determined based on the assumptions in Table 48. These assumptions were used in generating the general formula for SYS0001.

Object	Catalog Entries	Bytes	Bytes for Each Object	Pages for Each Object
Table	1 SYSCATALOG 1 SYSTABAUTH 2 SYSINDEXES	169 84 262	515	0.13
View	1 SYSCATALOG 1 SYSVIEWS 2 SYSTABAUTH 2 SYSUSAGE	169 200 168 102	639	0.16
Column	1 SYSCOLUMNS	156	156	0.04
Package	1 SYSPROGAUTH 15 SYSUSAGE	49 765	814	0.20
Dbpace (including package dbspaces)	1 SYSDBSACES	46	46	0.01
User	1 SYSUSERAUTH 50 SYSTABAUTH 50 SYSSYNONYMS 150 SYSCOLAUTH	35 4,200 1,800 10,800	16,835	4.11
Package dbspaces	255 SYSACCESS	13,260	13,260	3.24
Character Set	1 SYSCHARSETS	400	400	0.10
Keys	1 SYSKEYS 2 SYSKEYCOLS	89 134	223	0.05
Other	15 SYSOPTIONS	1200	1200	0.30

When a table is created, one entry is made in the SYSTEM.SYSCATALOG table and one in the SYSTEM.SYSTABAUTH table. This formula assumes an average of

two indexes for each table. For each index created, one entry is made in SYSTEM.SYSINDEXES.

When a view is created, one entry is made in SYSTEM.SYSCATALOG. In addition, as many as 32 entries are made in SYSTEM.SYSVIEWS. With the assumption that the average view definition is less than 254 bytes, only one row is required. One entry is also made in the SYSTEM.SYSTABAUTH and SYSTEM.SYSUSAGE tables for each table on which the view is defined. The general formula assumes that, on average, a view is defined on two tables.

One entry is made in SYSTEM.SYSCOLUMNS for every table and view column.

When a package is created, one entry is made in SYSTEM.SYSPROGAUTH. In addition, entries are made in SYSTEM.SYSUSAGE for every table, view, index, and dbspace used by the package. (A package uses a dbspace if it uses a table in the dbspace.)

The general formula assumes 15 such entries in SYSTEM.SYSUSAGE.

One entry is made in SYSTEM.SYSDBSPACES for each dbspace added to the database, including package dbspaces.

One entry is placed in SYSTEM.SYSUSERAUTH for each user of the database. Each user is assumed to have access to an average of 50 tables (and views) belonging to other users. This explains the 50 entries in SYSTEM.SYSTABAUTH and SYSTEM.SYSSYNONYMS. Specific column update authorization is assumed to average about 3 columns for each table (or view) that is shared (3 for each of the 50 tables or views). This yields an estimate of 150 entries in SYSTEM.SYSCOLAUTH for each user.

For each package dbspace added, one entry is made in SYSTEM.SYSDBSPACES, which was accounted for earlier, and 255 entries are made in SYSTEM.SYSACCESS. The 255 entries are made because all 255 packages are preallocated in the dbspace, even though they can all be empty.

For each character set you define, you must load one row into SYSTEM.SYSCHARSETS.

For each key, one row is added to SYSTEM.SYSKEYS, and two rows are added to SYSTEM.SYSKEYCOLS (assuming that each key is made up of two columns).

Finally, three rows exist in SYSTEM.SYSOPTIONS for every database.

Derivation of the General Formula for SYS0001 Storage Estimating

The assumptions in the preceding section provide a means of estimating the data pages required in SYS0001. Assuming the PCTFREE value for the SYS0001 dbspace is 0, the SYS0001 data pages are:

SYS0001 data pages = .13 x the number of tables
+ .16 x the number of views
+ .04 x the number of columns
+ .20 x the number of packages
+ .01 x the number of dbspaces
(including package dbspaces)
+ 4.11 x the number of users
+ 3.24 x the number of package dbspaces
+ .10 x the number of character sets
+ .05 x the number of keys
+ .30 (for the SYSTEM.SYSOPTIONS table)

To get the total number of SYS0001 dbspace pages, you must add the header pages and the index pages. SYS0001 has eight header pages. The initial set of catalog entries generated by the database generation process fills 4 pages. The PCTINDX value for SYS0001 is 60. Thus, to get the total number of pages you must add 12 and divide by 0.4:

SYS0001 pages = (12 + SYS0001 data pages) / 0.40

The SYS0001 data pages is your estimate for the number of data pages for your catalog entries.

Formula for SYS0001 Storage Estimating

When the adjustments described in “Derivation of the General Formula for SYS0001 Storage Estimating” on page 460 are made, the formula for the total number of SYS0001 dbspace pages becomes:

SYS0001 pages = 30 + .33 x the number of tables
+ .40 x the number of views
+ .10 x the number of columns
+ .50 x the number of packages
+ .03 x the number of dbspaces
(including package dbspaces)
+ 10.28 x the number of users
+ 8.10 x the number of package dbspaces
+ .25 x the number of character sets
+ .13 x the number of keys
(+ .74 (for the SYSTEM.SYSOPTIONS table))

This number should be rounded up to the next higher multiple of 128. Because the number of pages needed for the SYSTEM.SYSOPTIONS catalog table is so small, the number is omitted from the general formula and any further calculations.

Examples of Using the SYS0001 Storage Estimating Formula

The following examples illustrate the use of the general formula for estimating the required dbspace size for SYS0001.

For a Test Database

Table 49 illustrates the estimate for a small set of catalog tables that can be used in generating a test database.

Example Number of Objects	Number of Pages Calculation	Number of Pages
Reserved	30	30
50 tables	.33 X 50	17
100 views	.40 X 100	40
1500 columns	.10 X 1 500	150
25 packages	.50 X 25	13
50 dbspaces	.03 X 50	2
15 users	10.28 X 15	154
1 package dbspace	8.10 X 1	8
2 character sets	.25 X 2	1
20 keys	.13 X 20	3
Total number of SYS0001 pages = 418		
Rounded to the next higher multiple of 128 is: 512		

For an Application Development Database

Table 50 illustrates the estimate for a medium sized set of catalog tables that might be used in generating a test database to support development of multiple application systems. The number of package dbspaces needed was determined by adding the number of views to the number of packages and dividing the sum by 255. The maximum number of packages that can be defined in a package dbspace is 255. This number could be reduced if the packages are large. The maximum 255 packages may not fit in the allocated pages for the dbspace.

Example Number of Objects	Number of Pages Calculation	Number of Pages
Reserved	30	30
500 tables	.33 X 500	165
1000 views	.40 X 1000	400
15,000 columns	.10 X 15,000	1,500
50 packages	.50 X 50	25
500 dbspaces	.03 X 500	15
25 users	10.28 X 25	257
5 package dbspaces	8.10 X 5	40
6 character sets	.25 X 6	2
200 keys	.13 X 200	26
Total number of SYS0001 pages = 2461		
Rounded to the next higher multiple of 128 is: 2560		

For a Production Database

Table 51 illustrates the estimate for a large sized set of catalog tables that could be used to support a production database.

<i>Table 51. Example of Estimating the Catalog Dbspace for a Production Database</i>		
Example Number of Objects	Number of Pages Calculation	Number of Pages
Reserved	30	30
3000 tables	.33 X 3000	990
5000 views	.40 X 5000	2000
75,000 columns	.10 X 75,000	7500
250 packages	.50 X 250	125
500 dbspaces	.03 X 500	15
50 users	10.28 X 50	514
21 package dbspaces	8.10 X 21	170
6 character sets	.25 X 6	2
1,200 keys	.13 X 1 200	156
Total number of SYS0001 pages = 11,502		
Rounded to the next higher multiple of 128 is: 11,520		

Modifying the SYS0001 Storage Estimating General Formula

Table 52 on page 464 and Table 53 on page 465 assist you if you want to modify any of the assumptions used in deriving the general formula. If you have generated the starter database, you should compare the data in the catalog tables against the assumptions made here. You can do so by issuing UPDATE STATISTICS for each of the catalog tables after you have used the starter database. Queries against SYSTEM.SYSCATALOG give you the statistics for comparison.

Table 52. Your Estimated Stored Lengths of Catalog Rows

Catalog Table	Minimum Length	Maximum Length	Estimated Average Length
SYSACCESS	46	64	
SYSCATALOG	64	384	
SYSCCSIDS	39	39	39
SYSCHARSETS	393	411	
SYSCOLAUTH	46	82	
SYSCOLSTATS	27	123	
SYSCOLUMNS	56	400	
SYSDBSPACES	40	58	
SYSDROP	13	13	13
SYSINDEXES	62	232	
SYSKEYCOLS	55	91	
SYSKEYS	77	113	
SYSOPTIONS	13	303	
SYSPARMS	82	82	
SYSPROGAUTH	46	54	
SYSPSERVERS	11	281	
SYSROUTINES	58	581	
SYSSTRINGS	286	286	286
SYSSYNONYMS	26	62	
SYSTABAUTH	57	101	
SYSUSAGE	36	72	
SYSUSERAUTH	35	35	35
SYSVIEWS	20	292	

Table 53. Your Assumptions of Catalog Bytes or Pages for Each Object				
Object	Catalog Entries	Bytes	Bytes for Each Object	Pages for Each Object
Table	1 SYSCATALOG 1 SYSTABAUTH __ SYSINDEXES	___ ___ ___		
View	1 SYSCATALOG 1 SYSVIEWS __ SYSTABAUTH __ SYSUSAGE	___ ___ ___ ___		
Column	1 SYSCOLUMNS	___		
Package	1 SYSPROGAUTH __ SYSUSAGE	___ ___		
Dbpace (including package dbspaces)	1 SYSDBSACES	___		
User	1 SYSUSERAUTH __ SYSTABAUTH __ SYSSYNONYMS __ SYSCOLAUTH	35 ___ ___ ___		
Package dbspaces	255 SYSACCESS	___		
Character Set	1 SYSCHARSETS	___		
Keys	1 SYSKEYS __ SYSKEYCOLS	___ ___		
Other	3 SYSOPTIONS	___		

Estimating ISQL Dbpace Requirements

An allocation of 1 024 pages should be sufficient for most ISQL users. If you have many users or expect to make extensive use of the ISQL stored queries facility, consider increasing this allocation.

The recommended size (in pages) for the PUBLIC.ISQL dbspace is 1024 or .88 x the number of stored queries, whichever is larger.

Estimating Dbpace Sizes for Routines

The size of ROUTINE tables can vary greatly from user to user and from installation to installation. You can place the ROUTINE tables for all users in the same public dbspace, or you can place the ROUTINE table for each user in that user's private dbspace.

The following formulas are condensed versions of size estimation formulas in the *DB2 Server for VM Database Administration* manual. They simplify the work

required to estimate the size of the dbspace needed to hold routines. The following assumptions have been used in the formulas:

- The PCTFREE value is 15.
- The PCTINDEX value is 33.
- ALLOWANCE was not included in the formula.

The following formula can be used to calculate the average row length for the routines:

$$\text{AVGROWLEN} = 23 + \text{average command line length} + \text{average remark length}$$

Note: The average command line length is not the same as the average command length. A command can be entered on multiple command lines. A command line in a routine has a maximum length of 254 characters. A command has a maximum length of 2 048 characters. Be sure to use the command line length in your estimate.

The following formula can be used to calculate the number of dbspace pages required for your ROUTINE tables:

$$\frac{\text{number of users} \times \text{average number of routines} \times \text{average number of lines}}$$

$$\text{Dbspace pages} = (2074 / \text{AVGROWLEN}) \text{ (from previous formula)}$$

Examples:

Number of Users	Number of Routines For Each User	Number of Lines For Each Routine	Row Length	Dbspace Pages
1	20	20	80	16
20	20	20	80	309
50	60	35	75	3797

Estimating Dbspace Size for Stored SQL Statements (Stored Queries)

The following assumptions are used in the formula for calculating the size of the dbspace required for stored SQL statements:

- The PCTFREE value is 15.
- The PCTINDEX value is 33.
- ALLOWANCE is not included in the formula.
- One page was included for one routine named PROFILE. It can contain up to 25 lines with an average row length of 80.

The following formula can be used to calculate the number of dbspace pages needed for stored SQL statements:

$$\frac{\text{Dbspace pages} = 1 + (.037 \times \text{number of statements}) + ((\text{Truncate} [(\text{avglen} + 250) / 250 \times 250]) \times \text{number of statements})}{2667}$$

When calculating the average length of your stored queries, you must include the FORMAT information for all SELECT statements. The length of the FORMAT information can be calculated by the following formula:

$$\text{Format length} = 504 + (\text{number of columns} \times 44)$$

or

$$2048, \text{ whichever is smaller}$$

The following examples show the number of dbspace pages required for each user for the two types of stored SQL statements. The two types are:

- SQL SELECT statements (true stored queries)
- Other SQL statements.

The examples in Table 55 and Table 56 on page 468 show the number of dbspace pages required for one user for 10 stored SQL statements. If a user has some of each type of stored SQL statement, you must add the values from each table as needed.

Table 55. Examples — Dbspace Pages for Each User for Stored SELECT Statements

Number of Selects	Number of Columns	Length of Select	Format Length	Adjusted Length ¹	Dbspace Pages for Each User
10	10	70	944	1250	5.88
10	20	70	1384	1500	5.99
10	10	400	944	1500	5.99
10	20	400	1384	2000	7.87
10	40	400	2048	2500	9.38
10	46	2048	2048	4250	16.30

¹ The adjusted length is the stored data length of the stored SQL statements. For more information on adjusted lengths columns, see the *DB2 Server for VM Database Administration* manual.

Table 56. Examples — Dbspace Pages for Each User for Stored SQL Statements Other than SELECTs

Number of Commands	Average Length	Adjusted Length¹	Pages for Each User
10	70	250	1.31
10	400	500	2.25
10	999	1 000	4.12
10	1 499	1500	6.00
10	2 048	2 250	8.81

¹The adjusted length is the stored data length of the stored SQL statements. For more information on adjusted lengths columns, see the *DB2 Server for VM Database Administration* manual.

Appendix C. Maximum Values

Database Manager Maximum Values

<i>Table 57. Database Manager Maximum Values</i>	
Restricted Parameter	Maximum
Databases for each system ¹	unlimited
Number of communications links	65535
Initialization parameters:	
DISPBIAS	10
NCUSERS ²	252
NPACKAGE ²	32766
NPACKPCT	100
NPAGBUF ^{2 3}	400000
NDIRBUF ^{2 4}	400000
NLRBU ²	583333
NLRBS ²	583333
NCSCANS ²	655
CHKINTVL ²	99999999
SLOGCUSH	90
ARCHPCT	99
SOSLEVEL	100

Notes for Table 57:

1. A database machine can access only one database at a time. Many database machines can operate concurrently.
2. This is the absolute maximum value. The practical maximum value is less, depending on the values specified for other parameters and system resources (such as the amount of storage available). The maximum number of NCUSERS is limited because the program stack storage for each real agent is obtained below 16 megabytes.
3. These are 4 K pages.
4. These are 512-byte pages.

Database Maximum Values

<i>Table 58. Database Maximum Values</i>	
Restricted Parameter	Maximum
Number of storage pools	999
Number of dbextents ¹	999
Number of dbspaces	32000
Number of bytes per database ²	64 gigabytes ⁴
Number of pages per database ²	16,777,088
Number of bytes per dbspace ³	32 gigabytes ⁴
Number of pages per dbspace ³	8,388,480
Size of the directory	1 volume
Size of a log ⁵	524,287 4KB pages
Size of a dbextent	1 volume
Size of a storage pool ²	64 gigabytes ⁴

Notes for Table 58:

1. The database extents are not limited to 26 minidisks because VM block I/O is used.
2. This is the absolute maximum size of the database. The practical maximum is lower.
3. This is the absolute maximum size of a dbspace. The practical maximum is lower.
4. A gigabyte is 2³⁰ bytes (1,073,741,824).
5. This is the absolute maximum size allowed, but it is much larger than the database can use. See Table 3 on page 17 for more appropriate estimates.

Appendix D. Updating SYSTEM.SYSSTRINGS

The SYSTEM.SYSSTRINGS catalog table contains information on all the CCSID conversions that this product supports. For each CCSID conversion performed, there must be a corresponding row in this table.

To insert a row, follow these steps:

1. Determine the source and target CCSIDs
2. Determine the conversion type

The conversion type is based on the **encoding scheme** (EBCDIC or ASCII) of the CCSIDs and whether they are for tagging SBCS, mixed, or graphic data. Note that in any conversion that this product supports, the target CCSID is always EBCDIC. The following are conversion types recognized:

- 'SS' (EBCDIC/ASCII SBCS to EBCDIC SBCS)
- 'SM' (EBCDIC/ASCII SBCS to EBCDIC mixed)
- 'MS' (EBCDIC mixed to EBCDIC SBCS)
- 'MM' (EBCDIC mixed to EBCDIC mixed)
- 'PS' (ASCII mixed to EBCDIC SBCS)
- 'PM' (ASCII mixed to EBCDIC mixed)
- 'GG' (EBCDIC/ASCII graphic to EBCDIC graphic)

3. Determine the error byte

The error byte is only used for SBCS conversions, and therefore applies to all conversion types except for 'GG'. Be careful what you set it to: if a character in the source gets converted to the error byte, the conversion is terminated and an error occurs. CCSID conversions either have a NULL error byte, or are set as follows for the detection of DBCS characters in the source when they are not allowed:

- X'0E' used for
 - 'SM' conversions where the source CCSID is EBCDIC
 - 'MS' conversions
- X'3E' used for
 - 'SS' conversions where the source CCSID is ASCII
 - 'SM' conversions where the source CCSID is ASCII
 - 'PS' conversions

CDRA SBCS conversion tables are modified for use in this type of conversion, so that all DBCS first bytes get mapped to X'3E' instead of the original X'3F'.

For more information, see step 7, "Customize the SBCS Conversion Table."

4. Determine the substitution byte

As with the error byte, the substitution byte is not applicable in 'GG' type conversions. Whenever a character in the source gets converted to the substitution byte, warning flags are set in the SQLCA. This byte is set based on whether a given conversion table is created using the **enforced subset match** criterion. (For more on this subject, refer to the *Character Data Representation Architecture Level 1, Registry* manual.) If it is, then this byte is set to X'3F',

which is the CDRA-defined **SUB** character for conversions with EBCDIC target CCSIDs.

5. Determine the TRANSPROC

This field only applies in the cases of 'MM', 'PM', and 'GG' type conversions. It contains the name of the DBCS conversion table that is shipped with this product for use in the conversion. If it contains a value other than any of the DBCS conversion table names that the database manager recognizes, then the value is treated as the name of a user-defined DBCS conversion exit. (For information on creating a user-defined TRANSPROC exit, see "Coding Your Own TRANSPROC Exit" on page 380.)

6. Create the SBCS conversion table

An SBCS conversion table may be required except in the case of the 'GG' conversion type, where it is not applicable. For a conversion type where SBCS conversion applies, it is possible to specify a NULL SBCS conversion table. For example, you can have a mixed-to-mixed conversion where the SBCS CCSIDs of the source and target mixed CCSIDs are the same, in which case you do not need to perform conversion on the SBCS portion(s) of the mixed source data.

If you require a non-NULL SBCS conversion table, check the catalog table SYSTEM.SYSSTRINGS to see whether it is already supported. If it is not currently supported, then you have to create the conversion table based on the conversion mapping that you define.

A user-created SBCS conversion table should be in the same format as those supplied by CDRA: that is, a 256-byte string where the byte at offset *n* (starting at offset 0) corresponds to what codepoint *n* in the source CCSID is converted to. You also have to customize the conversion table for use if its source CCSID is ASCII SBCS or ASCII mixed.

7. Customize the SBCS conversion table

If your SBCS conversion table is to be used for a conversion with an ASCII SBCS or ASCII mixed source CCSID, you will have to modify it for the proper detection of DBCS first bytes. This requires that you determine the ranges of valid DBCS first byte codepoints for the ASCII SBCS source CCSID; then set the contents of the SBCS conversion table at the offsets that correspond to the DBCS first byte codepoints to:

a. Error byte

This is required for the following conversions, where DBCS characters are not allowed in the ASCII source:

- 1) 'SS' conversions where the source CCSID is ASCII
- 2) 'SM' conversions where the source CCSID is ASCII
- 3) 'PS' conversions

In the case of the CDRA-supplied SBCS conversion tables that are shipped for use in the types of conversion mentioned above, the values contained at the DBCS first byte offsets in the conversion tables have been changed from the original X'3F' to X'3E'. This is also the error byte for these conversions. X'3F' remains the substitution byte for these conversions.

You should also set the DBCS first byte offsets in your conversion table to a unique character, which will also be your error byte.

b. X'00'

This is required for 'PM' conversions, where DBCS characters in the ASCII source are allowed and are converted. The database manager considers a byte a DBCS first byte if it gets converted to X'00' using the ASCII-to-EBCDIC SBCS conversion table, and if it is not X'00' itself to begin with.

In the case of the CDRA-supplied SBCS conversion tables that are shipped for use in 'PM' conversions, the values contained at the DBCS first byte offsets in the conversion tables used to be X'3F', and have been changed to X'00'. You **must** therefore also set the characters at the DBCS first byte offsets in the conversion table to X'00', in order for DBCS characters to be recognized in the mixed source.

8. Insert the row into SYSTEM.SYSSTRINGS

You can create a DBSU job to insert the row into SYSTEM.SYSSTRINGS. For examples, review the ARITPOP MACRO which is supplied with the DB2 Server for VM code. This macro inserts a row into SYSTEM.SYSSTRINGS for every supported conversion between CCSIDs that are supplied with the database manager.

Appendix E. Defining Your Own Character Set

If you cannot use one of the IBM-supplied sample character sets, you can create your own. (You may be able to use one of the character sets in the *Character Data Representation Architecture Level 1, Registry* manual. The CCSIDs in this manual are registered, and already defined.) To create a character set:

1. Identify all characters in your set and their hexadecimal values. For more information, see “Step 1: Identify All Characters in Your Character Set” on page 476.

2. Classify each character.

The database manager has 12 character classifications that it uses to identify how the character can be used in SQL statements. You must classify those characters in your set that differ from the ENGLISH character set table. For more information, see “Step 2: Classify the Characters” on page 477.

3. Determine the hexadecimal values to which lowercase characters are to be translated. For example, suppose X'6A' is used for lowercase n-tilde, and X'7B' is used for uppercase N-tilde. The database manager does not automatically know that X'6A' should be converted to X'7B'. You must define that relationship. For more information, see “Step 3: Determine Translation Characters” on page 486.

4. Update the SYSTEM.SYSCHARSETS catalog table.

Having defined character classifications and translations, you must implement them on the database manager. You can code an INSERT command for the DBS utility to process. During this process, you must also choose a name for the character set (for example, PORTUGUESE). For more information, see “Step 4: Update the SYSTEM.SYSCHARSETS Catalog Table” on page 487.

5. Update the SYSTEM.SYSCCSIDS catalog table.

When the new character set is implemented, you must add a row to the SYSTEM.SYSCCSIDS catalog table to identify the CCSID values to be associated with the new character set. For more information, see “Step 5: Update the SYSTEM.SYSCCSIDS Catalog Table” on page 488.

6. Update the SYSTEM.SYSSTRINGS catalog table.

After you specify the CCSID values, you must indicate the conversion table information to allow conversions to and from the new CCSID. For more information, see “Step 6: Update the SYSTEM.SYSSTRINGS Catalog Table” on page 489.

7. Update the CCSID-Related CMS Files

You must now use the CCSID option of the ARISDBMA EXEC to load the CMS files to be able to use the new character set. This EXEC copies the information in SYSTEM.SYSCHARSETS, SYSTEM.SYSCCSIDS and SYSTEM.SYSSTRINGS to three CMS files. See the *DB2 Server for VM Program Directory* for more information on the ARISDBMA EXEC.

When your character set is loaded into the SYSTEM.SYSCHARSETS catalog table, and the SYSTEM.SYSCCSIDS and SYSTEM.SYSSTRINGS catalog tables have been updated, you can specify the character set by using the CHARNAME initialization parameter.

Each step in defining a character set is discussed below. As an example, a character set (PORTUGUESE) for Brazilian Portuguese is defined.

Step 1: Identify All Characters in Your Character Set

Identify all characters in your character set, and write a matrix like the one shown in the previous figures. For example, Figure 124 shows an example of a character set that might be used to represent Brazilian Portuguese.

		00				01				10				11				Bits 0,1	
		00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11		2,3
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
Bits 4567	Hex 1																		
0000	0					SP	&	-								õ	é		0
0001	1							/		a	j					A	J		1
0010	2									b	k	s				B	K	S	2
0011	3									c	l	t				C	L	T	3
0100	4									d	m	u				D	M	U	4
0101	5									e	n	v				E	N	V	5
0110	6									f	o	w				F	O	W	6
0111	7									g	p	x				G	P	X	7
1000	8									h	q	y				H	Q	Y	8
1001	9								ã	i	r	z				I	R	Z	9
1010	A					É	ç	ç	:						-				
1011	B					.	Ç	,	õ										
1100	C					<	*	%	Ã										
1101	D					()	_	'										
1110	E					+	:	>	=										
1111	F					!	^	?	"										

Figure 124. PORTUGUESE Character Set

Figure 125 on page 477 is provided for you to record your own character set.

	4	5	6	7	8	9	A	B	C	D	E	F
0												
1												
2												
3												

4												
5												
6												
7												
8												
9												
A												
B												
C												
D												
E												
F												

Figure 125. Your SBCS Character Set

At this time, you should note the hexadecimal values in your character set that have representations different from those in the ENGLISH character set in Figure 100 on page 324. Recording the differences makes character classification easier. These are the hexadecimal values that have different representations in the PORTUGUESE example:

```
X'4A'   X'5F'   X'7C'
X'4F'   X'6A'   X'C0'
X'5A'   X'79'   X'D0'
X'5B'   X'7B'
```

Step 2: Classify the Characters

When interpreting commands, the database manager must identify which characters are valid, and which are not. To do this, the database manager uses an internal character classification table.

In the table, each of the 256 possible SBCS hexadecimal values are assigned a classification. The database manager uses these classifications to tell whether a character is, for example, a delimiter or a numeric. There are 12 classes. Each hexadecimal value is assigned one of these classes. The only hexadecimal values that you are able to reclassify are those that, in the ENGLISH character set, are classified as 3 or 0. Values classified as 0 can be reclassified as 3, and values classified as 3 can be reclassified as 0. No other reclassifications are allowed. The only exception to this rule occurs with certain class 6 characters. See the class 6

description below for details. See Table 59 on page 482 for the ENGLISH character set class. Other character classes are shown only for reference:

Class Meaning

- 0** Unusable for keywords or unquoted identifiers
Any hexadecimal value assigned to this class cannot be used in keywords or unquoted identifiers.
- 1** Blank
The hexadecimal value assigned to this class is a blank. A blank, in the SQL language, is a delimiter between keywords. The database manager uses X'40' for blanks.
- 2** Apostrophe
The hexadecimal value assigned to this class is an apostrophe ('). An apostrophe, in the SQL language, is the delimiter for character constants. The database manager uses X'27' for an apostrophe.
- 3** Characters other than numerics, uppercase English alphabets, and underscores that are usable for unquoted identifiers
Numeric, uppercase English alphabets, and underscores all belong to other classes. In the default ENGLISH character set, the lowercase alphabets along with \$, #, and .* are assigned to this class. In the sample character sets, characters such as n-tilde and o-umlaut are assigned this class.
- 4** Numerics
Any hexadecimal value assigned to this class is a numeric. The SQL language defines the X'F0' to X'F9' to represent the numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. You must not assign class 4 to any other hexadecimal values, nor can you reassign hexadecimal values X'F0' to X'F9' to some other class.
- 5** Period
Any hexadecimal value assigned to this class is a period. A period, in the SQL language, is the delimiter between a qualifier (such as an owner) and a data object (such as a table). The database manager uses X'2E' for a period.
- 6** Special characters
Hexadecimal values assigned to this class have special meanings in the SQL language, just as numerics do. You must not assign class 6 to any hexadecimal values other than those listed below. Nor can you reassign the hexadecimal values shown to some other class. The only exceptions are the ones which have a different hexadecimal value depending on the application server default CCSID used. For those hexadecimal values listed which map to a character used in the SQL language for your application server default CCSID, do not reassign these values. For those hexadecimal values listed which do not map to a character used in the SQL language for your application server default CCSID, you can assign them to class 0 or class 3.

For example, X'5A' maps to the exclamation mark (!) for CCSID 37. For CCSID 500, X'5A' maps to the right square bracket (]). For CCSID 37, the

hexadecimal value should be class 6. For CCSID 500, the hexadecimal value could be either class 0 or class 3. In the SQL language the following hex values have these meanings:

X'4C' <
X'4D' (
X'4E' +
X'4F' | (for CCSIDs 37, 284, 285, 290, 420, 424, 833, 836, 838, 1027, 28709)
X'4F' ! (for CCSIDs 273, 277, 278, 280, 297, 500, 870, 871, 875)
X'50' &
X'5A' ! (for CCSIDs 37, 285, 290, 420, 424, 833, 836, 838, 1027, 28709)
X'5C' *
X'5D')
X'5E' ;
X'5F' ~ (for CCSIDs 37, 284, 285, 290, 420, 424, 833, 836, 838, 1027, 28709)
X'5F' ^ (for CCSIDs 273, 277, 278, 280, 297, 500, 870, 875)
X'60' -
X'61' /
X'69' ^ (for CCSID 838)
X'6A' | (for CCSIDs 870, 878)
X'6B' ,
X'6C' %
X'6E' >
X'6F' ?
X'7A' :
X'7E' =
X'B0' ^ (for CCSIDs 37, 290, 424, 833, 836, 1027, 28709)
X'BA' ~ (for CCSIDs 273, 277, 278, 280, 297, 500, 871)
X'BA' ^ (for CCSIDs 284, 285)
X'BB' | (for CCSIDs 273, 277, 278, 280, 297, 500, 871)
X'BB' ! (for CCSID 284)
X'EC' ^ (for CCSID 871)
X'EF' ~ (for CCSID 875)

7 Quotation Mark

Any hexadecimal value assigned to this class is a double quotation mark ("). A double quotation mark, in the SQL language, is the delimiter for quoted identifiers. The database manager uses X'7F' for a double quotation mark.

8 Shift-out character

You should not assign any hexadecimal value to this class. When the DBCS option is YES, the database manager assigns this class to X'0E'.

9 Shift-in character

You should not assign any hexadecimal value to this class. When the DBCS option is YES, the database manager assigns this class to X'0F'.

A English Uppercase Alphabets

This class is restricted to all English uppercase alphabets (hexadecimal values X'C1' to X'C9', X'D1' to X'D9', and X'E2' to X'E9'). English uppercase alphabets can be used in unquoted identifiers and keywords. (This is true no matter what SBCS character set is specified.)

B Underscore

Any hexadecimal value assigned to this class is an underscore. An underscore, in the SQL language, can be used in an unquoted identifier except as a starting character. The database manager uses X'6D' for an underscore.

When you have defined a character set, you must classify each hexadecimal value that has a different representation in your character set than it does in the ENGLISH character set.

The database manager always sets the first 64 hexadecimal values (X'00' to X'3F') to class 0. You can set only the remaining 192 hexadecimal values. Therefore, if any character in your set has a hexadecimal value within X'00' to X'3F', you can use that hexadecimal value only in quoted identifiers.

The only hexadecimal values that the database manager reclassifies in the first 64 are X'0E' and X'0F'. Those hexadecimal values are permanently defined to the database manager as the DBCS shift-out and shift-in characters. When the DBCS option is YES, the database manager reclassifies X'0E' to class 8 and X'0F' to class 9. For more information, see "Using Double-Byte Character Set (DBCS)" on page 331.

Not all SBCS character sets can be classified for use with the database manager, because it reserves certain hexadecimal values. For example, all hexadecimal values that (in the ENGLISH character set) represent uppercase English letters are reserved. The database manager reserves hexadecimal values so it can correctly interpret SQL statements.

Use Table 59 on page 482 to classify your character set. The first column gives the hexadecimal value. The next two columns identify the ENGLISH classification and conversion values for each of those hexadecimal values. (Translation values are discussed in the next step.) The fourth and fifth columns show the classification and conversion values for the PORTUGUESE example. The remaining two columns are for your own character set.

Note: All hexadecimal values are reserved except those that are classified as 0 or 3 in the ENGLISH character set.

Any hexadecimal value that is classified in the ENGLISH character set as 0 or 3 can be reclassified as 3 or 0. Keep in mind that all hexadecimal values that are classified as 0 cannot be used in keywords and unquoted identifiers. Therefore, you

would not want to classify as 0 any letter that is within your language's alphabet. You would not be able to use those letters in unquoted identifiers.

The English alphabet consists of the following letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, and Z. Most likely, the hexadecimal values for letters in your language that are not in the English alphabet are classified as 0 in the ENGLISH character set. You would typically change the classification to 3.

If you must reclassify a hexadecimal value, but the hexadecimal value is reserved, then it is not possible to completely classify the character set. In that situation, it may not be to your advantage to specify an alternative character set. For example, if a character in your alphabet has a hexadecimal value that is assigned class 6 in the ENGLISH character set, you cannot reclassify that hexadecimal value (the only exceptions are the hexadecimal values associated with the characters |, !, ~ and ^).

The rationale used in classifying the PORTUGUESE character set hexadecimal values that are different from the ENGLISH character set is as follows:

- X'4A'** Because the character represented by this hexadecimal value is in the Portuguese alphabet, the hexadecimal value was reclassified from 0 to 3.
- X'4F'** The character represented by this hexadecimal value in the PORTUGUESE character set is the exclamation mark. Since this is also a special character in the ENGLISH character set and is already classified as 6, there is no need to reclassify it.
- X'5A'** This hexadecimal value, which represents a dollar sign (\$) in the PORTUGUESE character set, was reclassified from 6 to 0. This was done because the dollar sign is not a special character in the SQL language. If you want to use the dollar sign in unquoted identifiers and keywords, however, you can reclassify it to 3.
- X'5B'** Because the character represented by this hexadecimal value is in the Portuguese alphabet, the hexadecimal value should be classified as 3. It is already classified as a 3 in the ENGLISH classifications, so there is no need to reclassify it.
- X'5F'** This is another character that is reserved in the SQL language. In the example PORTUGUESE character set, X'5F' represents a caret. Since this is also a special character in the ENGLISH character set and is already classified as 6, there is no need to reclassify it.
- X'6A'** Because the character represented by this hexadecimal value is in the Portuguese alphabet, the hexadecimal value was reclassified from 0 to 3.
- X'79'** Because the character represented by this hexadecimal value is in the Portuguese alphabet, the hexadecimal value was reclassified from 0 to 3.
- X'7B'** Because the character represented by this hexadecimal value is in the Portuguese alphabet, the hexadecimal value should be classified as 3. It is already classified as a 3 in the ENGLISH classifications, so there is no need to reclassify it.
- X'7C'** Because the character represented by this hexadecimal value is in the Portuguese alphabet, the hexadecimal value should be classified as 3. It is already classified as a 3 in the ENGLISH classifications, so there is no need to reclassify it.

- X'B0'** This hexadecimal value, which represents a cent sign (¢) in the PORTUGUESE character set, was reclassified from 6 to 0. This was done because the cent sign is not a special character in the SQL language.
- X'BA'** The character represented by this hexadecimal value in the PORTUGUESE character set is the NOT sign. Since this is a special character in the SQL language, the value was reclassified from 0 to 6.
- X'BB'** The character represented by this hexadecimal value in the PORTUGUESE character set is the vertical bar. Since this is a special character in the SQL language, the value was reclassified from 0 to 6.
- X'C0'** Because the character represented by this hexadecimal value is in the Portuguese alphabet, the hexadecimal value was reclassified from 0 to 3.
- X'D0'** Because the character represented by this hexadecimal value is in the Portuguese alphabet, the hexadecimal value was reclassified from 0 to 3.

Having reclassified the characters, you next need to consider the translation values of those characters.

Table 59 (Page 1 of 5). Character Classification and Translation Table

Hex Value	English Class.	English Trans.	Brazilian Class.	Brazilian Trans.	Your Class.	Your Trans.
40	1	40				
41	0	41				
42	0	42				
43	0	43				
44	0	44				
45	0	45				
46	0	46				
47	0	47				
48	0	48				
49	0	49				
4A	0	4A	3			
4B	5	4B				
4C	6	4C				
4D	6	4D				
4E	6	4E				
4F	6	4F				
50	6	50				
51	0	51				
52	0	52				
53	0	53				
54	0	54				
55	0	55				
56	0	56				
57	0	57				
58	0	58				
59	0	59				
5A	6	5A	0			
5B	3	5B				
5C	6	5C				
5D	6	5D				
5E	6	5E				
5F	6	5F				

Table 59 (Page 2 of 5). Character Classification and Translation Table

Hex Value	English Class.	English Trans.	Brazilian Class.	Brazilian Trans.	Your Class.	Your Trans.
60	6	60				
61	6	61				
62	0	62				
63	0	63				
64	0	64				
65	0	65				
66	0	66				
67	0	67				
68	0	68				
69	0	69				
6A	0	6A	3	X'5B'		
6B	6	6B				
6C	6	6C				
6D	B	6D				
6E	6	6E				
6F	6	6F				
70	0	70				
71	0	71				
72	0	72				
73	0	73				
74	0	74				
75	0	75				
76	0	76				
77	0	77				
78	0	78				
79	0	79	3	X'7C'		
7A	6	7A				
7B	3	7B				
7C	3	7C				
7D	2	7D				
7E	6	7E				
7F	7	7F				
80	0	80				
81	3	C1				
82	3	C2				
83	3	C3				
84	3	C4				
85	3	C5				
86	3	C6				
87	3	C7				
88	3	C8				
89	3	C9				
8A	0	8A				
8B	0	8B				
8C	0	8C				
8D	0	8D				
8E	0	8E				
8F	0	8F				

Table 59 (Page 3 of 5). Character Classification and Translation Table

Hex Value	English Class.	English Trans.	Brazilian Class.	Brazilian Trans.	Your Class.	Your Trans.
90	0	90				
91	3	D1				
92	3	D2				
93	3	D3				
94	3	D4				
95	3	D5				
96	3	D6				
97	3	D7				
98	3	D8				
99	3	D9				
9A	0	9A				
9B	0	9B				
9C	0	9C				
9D	0	9D				
9E	0	9E				
9F	0	9F				
A0	0	A0				
A1	0	A1				
A2	3	E2				
A3	3	E3				
A4	3	E4				
A5	3	E5				
A6	3	E6				
A7	3	E7				
A8	3	E8				
A9	3	E9				
AA	0	AA				
AB	0	AB				
AC	0	AC				
AD	0	AD				
AE	0	AE				
AF	0	AF				
B0	6	B0	0			
B1	0	B1				
B2	0	B2				
B3	0	B3				
B4	0	B4				
B5	0	B5				
B6	0	B6				
B7	0	B7				
B8	0	B8				
B9	0	B9				
BA	0	BA	6			
BB	0	BB	6			
BC	0	BC				
BD	0	BD				
BE	0	BE				
BF	0	BF				

Table 59 (Page 4 of 5). Character Classification and Translation Table

Hex Value	English Class.	English Trans.	Brazilian Class.	Brazilian Trans.	Your Class.	Your Trans.
C0	0	C0	3	X'7B'		
C1	A	C1				
C2	A	C2				
C3	A	C3				
C4	A	C4				
C5	A	C5				
C6	A	C6				
C7	A	C7				
C8	A	C8				
C9	A	C9				
CA	0	CA				
CB	0	CB				
CC	0	CC				
CD	0	CD				
CE	0	CE				
CF	0	CF				
D0	0	D0	3	X'4A'		
D1	A	D1				
D2	A	D2				
D3	A	D3				
D4	A	D4				
D5	A	D5				
D6	A	D6				
D7	A	D7				
D8	A	D8				
D9	A	D9				
DA	0	DA				
DB	0	DB				
DC	0	DC				
DD	0	DD				
DE	0	DE				
DF	0	DF				
E0	0	E0				
E1	0	E1				
E2	A	E2				
E3	A	E3				
E4	A	E4				
E5	A	E5				
E6	A	E6				
E7	A	E7				
E8	A	E8				
E9	A	E9				
EA	0	EA				
EB	0	EB				
EC	0	EC				
ED	0	ED				
EE	0	EE				
EF	0	EF				

Table 59 (Page 5 of 5). Character Classification and Translation Table

Hex Value	English Class.	English Trans.	Brazilian Class.	Brazilian Trans.	Your Class.	Your Trans.
F0	4	F0				
F1	4	F1				
F2	4	F2				
F3	4	F3				
F4	4	F4				
F5	4	F5				
F6	4	F6				
F7	4	F7				
F8	4	F8				
F9	4	F9				
FA	0	FA				
FB	0	FB				
FC	0	FC				
FD	0	FD				
FE	0	FE				
FF	0	FF				

Step 3: Determine Translation Characters

When the database manager translates a character string from lowercase to uppercase, it checks the classification of each character in the string. If the character is in class 3, it is translated. If not, the character is not changed.

To translate the character, the database manager consults a translation table. The translation table contains the hexadecimal value to which a particular hexadecimal value is to be translated.

For every hexadecimal value in your set that has a different character representation than in English, you must define a translation value. Refer again to Table 59 on page 482. The following rationale was used to choose the PORTUGUESE translation values:

- X'4A'** Because the character represented by this hexadecimal value is an uppercase E with an accent, there is no need to translate the hexadecimal value to some other value when the database manager is folding to uppercase. The translation value remains X'4A' (the same as the ENGLISH value).
- X'4F'** The character represented by this hexadecimal value in Portuguese is the exclamation mark. Because the hexadecimal value should not be changed when the database manager is doing a lowercase to uppercase translation, the translation value should remain X'4F'.
- X'5A'** This hexadecimal value, which represents a dollar sign (\$) in the PORTUGUESE character set, should not change during a translation. The translation value should remain X'5A'.
- X'5B'** Because this hexadecimal value represents an uppercase C with a cedilla, it does not need to be changed during a translation. The translation value should remain X'5B'.
- X'5F'** This is another hexadecimal value that does not represent a letter of the alphabet. The hexadecimal value should not change during a translation. The translation value should remain X'5F'.

- X'6A'** This hexadecimal value represents a lowercase c with a cedilla. During a translation, it should be translated to an uppercase C with a cedilla. To have the database manager do the correct translation, the translation value should be X'5B' (the hexadecimal value for uppercase C with a cedilla).
- X'79'** This hexadecimal value represents a lowercase a with a tilde. During a translation, it should be translated to an uppercase A with a tilde. The translation value should be X'7C'.
- X'7B'** Because this hexadecimal value represents an uppercase O with a tilde, it does not need to be changed during a translation. The translation value should remain X'7B'.
- X'7C'** Because this hexadecimal value represents an uppercase A with a tilde, it does not need to be changed during a translation. The translation value should remain X'7C'.
- X'C0'** This hexadecimal value represents a lowercase o with a tilde. During a translation, it should be translated to an uppercase O with a tilde. To have the database manager do the correct translation, the translation value should be X'7B' (the hexadecimal value for uppercase O with a tilde).
- X'D0'** This hexadecimal value represents a lowercase e with an accent. During a translation, it should be translated to an uppercase E with an accent. To have the database manager do the correct translation, the translation value should be X'4A' (the hexadecimal value for uppercase E with an accent).

After determining what the translation values are, verify the following:

1. Hexadecimal values that you have reclassified to class 0 must be translated into identical hexadecimal values. If you reclassify X'A2' from 3 to 0, you must ensure that the translation value is set to X'A2', not X'E2' (as it is in ENGLISH). In the PORTUGUESE example, this situation did not occur. No hexadecimal values were reclassified from 3 to 0.
2. Hexadecimal values that you have reclassified to class 3 can be translated into any hexadecimal value having a class of 3 or A. A quick check of the PORTUGUESE-unique translation values show that the hexadecimal values either translate to themselves or to hexadecimal values having class 3. The PORTUGUESE example is still valid.

If your character set fails either of these tests, there is probably an error either in reclassifications or in the translation values chosen.

Step 4: Update the **SYSTEM.SYSCHARSETS** Catalog Table

After you define translation values for the characters that require them, load the character set into the **SYSTEM.SYSCHARSETS** catalog table. The easiest way to load a character set is by modifying a copy of the DBS utility control commands that load the sample ENGLISH character set. The file **ARISCHAR MACRO V** contains these control commands.

Change your copy of **ARISCHAR** to reflect the classification and translation values for your character set.

The first value in the **INSERT** statement is the name of the character set. For **'ENGLISH'** substitute the name of your character set. You can specify up to

eighteen characters. The value 'PORTUGUESE' was chosen as the name of the example Brazilian Portuguese SBCS character set.

The second value in the INSERT statement contains data for the character classification table. There are 192 character classifications that you can set. You should change only those character classifications in your character set that differ from the ENGLISH classifications. Use the values you have recorded in Table 59 on page 482.

The third value in the INSERT statement contains data for the character translation table. There are 192 character translation values you can set. You should change only those translation values in your character set that differ from the ENGLISH translation values. Use the values you have recorded in Table 59 on page 482. Note that the single quote (X'7D') must be entered twice. A single quote normally delimits the end of a value in an INSERT statement. To use a single quote as part of the data, the single quote must be entered twice.

Step 5: Update the SYSTEM.SYSCCSIDS Catalog Table

You must add a row to the SYSTEM.SYSCCSIDS catalog table to identify the CCSID values to be associated with your new character set. You could issue the following statement to update SYSTEM.SYSCCSIDS for the character set defined for this example:

```
INSERT INTO SYSTEM.SYSCCSIDS (CCSID,SUBTYPE,DBC SID,SBCSID,CHARNAME)
VALUES (57344,
       'S',
       0,
       0,
       'PORTUGUESE')
```

If you are defining your own CCSID (that is, one that is not obtained from the Character Data Representation Architecture (CDRA)) registry, you must use a value that is within the range of 57 344 to 61 439 (X'E000' to X'FFFF'). Values within this range are reserved for user-defined CCSIDs. Ensure that the value you specify does not already exist: the CCSID column cannot contain duplicate information. Also keep the following in mind:

- If the character set that you are defining uses conversion tables that are provided by the CDRA registry, use the CCSIDs that they indicate.
- The SUBTYPE column identifies the subtype of the CCSID. In this example, the value is 'S' for SBCS.
- The SBCSID column and the DBCSID column specify the SBCS and the DBCS components for a mixed CCSID. Because the CCSID in this example is SBCS, the value for both of these columns is 0.
- The value that you specify for the CHARNAME column must be the same as the value that you specified in the NAME column of the SYSTEM.SYSCCHARSETS catalog table.

For examples of statements that insert rows into the SYSTEM.SYSCCSIDS catalog table, review the ARITPOP MACRO that is supplied with the database manager.

Step 6: Update the SYSTEM.SYSSTRINGS Catalog Table

The SYSTEM.SYSSTRINGS catalog table identifies the tables that will be used for conversion between specific pairs of CCSIDs. Conversion tables for CDRA-supplied CCSIDs are provided by the CDRA registry. For more information on CDRA conversion tables, see the *Character Data Representation Architecture Level 1, Registry* manual. After you create your CCSID, you must determine the conversion table information for SYSTEM.SYSSTRINGS. You must add a row to SYSTEM.SYSSTRINGS for each conversion that you want to support both to and from the new CCSID. For a detailed description to update SYSTEM.SYSSTRINGS, see Appendix D, “Updating SYSTEM.SYSSTRINGS” on page 471.

Suppose you added CCSID 57344 and you want to support the following conversions:

- CCSID 37 to CCSID 57344
- CCSID 57344 to CCSID 37
- CCSID 57344 to CCSID 28709.

You must add three rows to the SYSTEM.SYSSTRINGS catalog table. To specify any of these conversions in SYSTEM.SYSSTRINGS, you would use an INSERT statement to insert the necessary information into the following columns of the catalog table:

- INCCSID, which specifies the CCSID of the input character.
- OUTCCSID, which specifies the CCSID to which the conversion is done.
- TRANSTYPE, which identifies the type of conversion to be done (for example, 'SS' for SBCS to SBCS).
- ERRORBYTE, which identifies characters that have no representation in the target code page. If a character to be converted maps to a code point containing this byte an error occurs.
- SUBBYTE, which identifies characters that have no representation in the target code page. If a character to be converted maps to a code point identified by this byte, a warning is issued.
- TRANSPROC, which identifies the conversion procedures that are used for conversion between CCSIDs. The procedures are used either for converting between DBCS CCSIDs, or for converting the DBCS components of mixed CCSIDs. The TRANSPROC value is blank if a DBCS conversion procedure is not applied. For more information, see “Coding Your Own TRANSPROC Exit” on page 380.
- TRANSTAB1, which represents the first 64 bytes of the conversion table.
- TRANSTAB2, which represents the last 192 bytes of the conversion table.

The conversion table maps the hexadecimal representation of each character in the source CCSID to the hexadecimal representation of each character in the target CCSID. For example, in CCSID 37 an exclamation mark (!) is represented by X'5A'. The hexadecimal representation for the exclamation mark in CCSID 281 is X'4F'. The hexadecimal value of the character at offset 90 in the conversion table where INCCSID=37 and OUTCCSID=281 would be X'4F'. Remember, when counting offsets, the *first* offset is *zero*. Therefore, the byte at offset 90 is actually the 91st.

For a detailed description to update SYSTEM.SYSSTRINGS, see Appendix D, “Updating SYSTEM.SYSSTRINGS” on page 471.

Step 7: Update the CCSID-Related CMS Files

After you have updated the SYSTEM.SYSCCHARSETS, SYSTEM.SYSCCSIDS and SYSTEM.SYSSTRINGS catalog tables, use the ARISDBMA EXEC to load the CCSID information from the catalog tables into three CMS files (ARISCCS, ARISSTR, and ARISSCR). The application server and the application requester use these files to enable the use of the new character set. For more information on the ARISDBMA EXEC, refer to the *DB2 Server for VM Program Directory*.

When you have completed steps 1 through 7, you can start the application server using the newly defined character set. If the database manager detects an error in the character set, it uses the value of CHARNAME that was used the last time the application server was started.

Appendix F. Macro List

The macros identified in this appendix are provided as programming interfaces for customers by the DB2 Server for VM database management system.

Attention: Do not use as programming interfaces any DB2 Server for VM macros other than those identified in this chapter.

Macro list

The database manager provides the following General-use programming interface macros:

- ARIRCAN
- ARIBFPPB.

Appendix G. Service and Maintenance Execs

This appendix describes the following execs:

- ARISAVES
- ARISPDFC
- SQLBOOTS
- SQLGENLD

ARISAVES EXEC

The VMSES/E VMFBLD exec calls ARISAVES to load and save each segment.

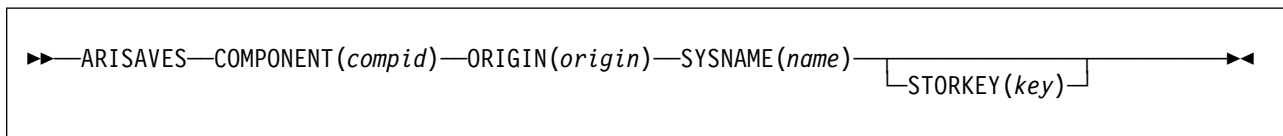


Figure 126. ARISAVES EXEC

COMPONENT(*compid*)

You must specify this parameter. It identifies the component for which a saved segment is generated. The valid values for *compid* are:

RA	For the resource adapter (RA), DRRM and CONV
LANGxxxx	For the national language message segment (where xxxx is the language key; LANGS001 is used for American mixed case English)
ISQL	For ISQL
DBSS	For DBSS and DSC
RDS	For RDS, WUM, DRRM and CONV

ORIGIN(*origin*)

You must specify this parameter. It identifies the saved segment load address, and is specified as a hexadecimal value. The ARISAVES EXEC does not check that the origins you specify give enough virtual storage to each component. However, if there is not enough room, the component is unusable.

Specify the origin that your System Programmer determined for the saved segment.

SYSNAME(*name*)

You must specify this parameter.

Specify the name you specified for the saved segment name in the OBJNAME field of the Add Segment Definition panel. See Chapter 8, “Saved Segments” on page 181. For example, SQLSQLDS for the DBSS component.

STORKEY(*key*)

This parameter is optional. For *key*, specify the storage key value (in decimal) for the saved segment.

If you omit this parameter, 13 is used for STORKEY. Normally, you would not specify this parameter.

VMSES/E Consideration

To use a different storage key, you need to change the build lists for the segments you are going to use. The following table lists the segment and their corresponding build list name.

Segment Name	Buildlist Name
SQLRMGR	ARIBLMGR
SQLISQL	ARIBLISQ
SQLSQLDS	ARIBLDBS
SQLXRDS	ARIBLRDS
LANGS001	ARIBLLNG

To change the build lists, you must use the VMSES/E local modification procedures described in the "Local Modifications for DB2 for VM" appendix in the *DB2 Server for VM Program Directory*.

End of VMSES/E Consideration

For example, to save the DBSS at origin B00000, invoke the ARISAVES EXEC as follows:

```
ARISAVES COMPONENT (DBSS) ORIGIN (B00000) SYSNAME (SQLSQLDS)
```

The ARISAVES EXEC does the following:

- Ensures the CP EMSG function is off.
- Reads the link book text file for the identified component.
- Verifies the text file exists.
- Prompts you asking whether the saved segments are to be the new default saved segments. If you reply YES, ARISAVES updates the origin values in the ARISSEGC MACRO, and generates bootstrap modules by calling the SQLBOOTS EXEC.

The ARISSEGC MACRO identifies the contents of the default (SQLDBA) bootstrap package. For more information, see "Step 12. Update the ARISSEGC Macro" on page 195.

If you reply NO, you must create a bootstrap package yourself by using the SQLGENLD EXEC shown in "Step 15. Create a Bootstrap Package" on page 198.

- Builds and runs a CMS LOAD command and subsequent CMS INCLUDE commands to create the saved segment at the specified origin address. All text files must reside on the service minidisk except for the text file ARIRVSTC, which must be on the production minidisk.
- Restores the original setting of the CP EMSG function.
- Prints the load map for the saved segment to a virtual print file.

- Issues a CMS SETKEY command to set the storage key for the saved segment.
- Issues a CP SAVESEG command to save the saved segment.

ARISPDFC EXEC

The ARISPDFC EXEC copies IBM-supplied production code from a source service disk or SFS directory (accessed as filemode V) to a target (secondary) production disk or directory. (Secondary production disks and directories are described in “Types of Database Machines” on page 284.)

You can use this EXEC to create additional primary database virtual machines. (A primary database machine owns a production minidisk.) You can also use it to service a secondary production minidisk. This is described in the *DB2 Server for VM Program Directory*.

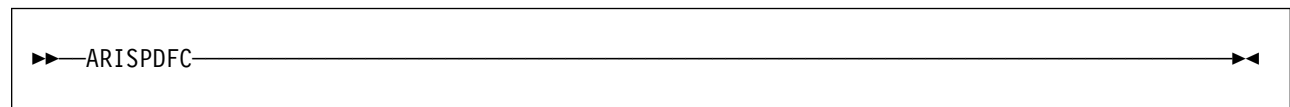
Authorization

To run the ARISPDFC EXEC, you must have:

- Read access to the source service minidisk or SFS directory (filemode V). You need to specify your default language with the SET LANG command after accessing this disk.
- Write access to the target production minidisk or SFS directory (file mode Q)
- CP CLASS = G.

Note: You can only issue this EXEC in the database machine that owns the target production disk or SFS directory.

Syntax



Description

The ARISPDFC EXEC does the following:

- Copies the files that make up DB2 Server for VM, identified by the contents of ARISPDEC MACRO V (except for the ARISPIDC MACRO, the ARISNLSC MACRO, and the ARISSEGC MACRO), from the source service minidisk or directory (filemode V) to the target production minidisk or directory (filemode Q).
- Copies the ARISQLLD LOADLIB Q file at the *latest service level*. (The “latest service level” includes the VM Data Spaces Support Feature or the DRDA code if it is currently installed on your source service disk or directory.)
- Erases (if it exists) and creates the CMS file ARISPIDC MACRO Q that identifies the source service disk or directory and this version of the target production disk or directory. The first record identifies the target; the second identifies the source.

Notes:

After running the ARISPDFC EXEC, you should:

1. Copy the ARISNLSC MACRO and ARISSEGC MACRO to your target (secondary) production disk or directory from the primary production disk or directory.

2. Acquire read access to your target production minidisk by entering:

```
RELEASE Q (DETACH
LINK machid cuu cuu RR
ACCESS cuu Q
```

If you are using SFS, acquire read access to SFS production directory by entering:

```
ACCESS machid.SQL.PRODUCTION Q
```

SQLBOOTS EXEC

The SQLBOOTS EXEC creates the default bootstrap files that must exist on the DB2 Server for VM production minidisk or SFS directory.

Authorization

To run the SQLBOOTS EXEC, you must have:

- Read access to the service minidisk or SFS directory
- Write access to the production minidisk or SFS directory.

Also ensure that the database machine using the production minidisk or SFS production directory to be updated is not active.

Syntax

```
▶▶—SQLBOOTS—▶▶
```

Description

The SQLBOOTS EXEC creates the following bootstrap files:

DBSS/RDS	SQLDBA SQLDBBT Q
ISQL	SQLDBA SQLISBT Q
Resource Adapter	SQLDBA SQLRMBT Q

SQLGENLD EXEC

The SQLGENLD EXEC recreates the bootstrap packages in saved segments.

Authorization

To run the SQLGENLD EXEC, you must have:

- Read access to the service minidisk or SFS directory
- If you run this EXEC from a database machine that does not have write access to the production minidisk or SFS directory, the EXEC will ask for the write password.

You must only run this EXEC from a database machine (single user mode).

Syntax

```
▶▶—SQLGENLD—◀◀
```

Description

The SQLGENLD EXEC creates the following bootstrap packages:

DBSS/RDS *dcssid* SQLDBBT Q

ISQL *dcssid* SQLISBT Q

Resource Adapter *dcssid* SQLRMBT Q

Above, *dcssid* is the name for the new bootstrap package.

For more information on this EXEC, see “Step 15. Create a Bootstrap Package” on page 198.

Appendix H. DRDA Considerations

Users who are planning to design applications that:

- run on non-VM platforms and use the Distributed Relation Database Architecture (DRDA) protocol to connect to DB2 for VSE & VM servers, or
- run on VM/ESA and use the Distributed Relation Database Architecture (DRDA) protocol to connect to DB2 family servers other than DB2 for VSE & VM

need to be aware that DB2 for VSE and VM's support of SQL does not exactly match the IBM SQL standard⁵ or the SQL Entry Level standard.⁶

This appendix attempts to provide some guidance in discrepancies to these standards.

Omissions from the Standards

For a list of where DB2 for VSE & VM does not support the IBM SQL or SQL92 entry level standards, please consult the *DB2 Server for VSE & VM SQL Reference manual*.

Extensions to the Standards

1. VM applications may not use the "userid IDENTIFIED BY password" clause on the CONNECT statement when using the DRDA protocol.
2. Packages that were created in SQLDS protocol by using extended dynamic statements⁷ cannot be processed in DRDA protocol, or the other way around.
3. There is no support for modifiable packages created by using extended dynamic statements. If you request such support by specifying the MODIFY option on the CREATE PACKAGE statement, the system will override this option with NOMODIFY.
4. Nonmodifiable packages created by using extended dynamic statements are supported with the following restrictions:
 - a. There is no support for the positioned UPDATE and positioned DELETE statements.
 - b. If the Basic Extended PREPARE form of the extended PREPARE statement prepares a statement that contains parameter markers, the USING DESCRIPTOR clause must be used to identify an input SQLDA structure.
 - c. There is no support for the Single Row Extended PREPARE form of the extended PREPARE statement.
 - d. There is no support for the NODESCRIBE option of the CREATE PACKAGE statement. If specified, it will be ignored.

⁵ IBM SQL is a superset of the SQL92 Entry Level standard

⁶ Entry Level of the International Organization for Standardization (ISO) 9075-1992 Database Language SQL specification

⁷ Since DB2 RXSQL uses extended dynamic statements, any restrictions on the use of extended dynamics apply to DB2 RXSQL as well.

- e. There is no support for “USER” in the ISOLATION option of the CREATE PACKAGE statement. The system will override USER with CS.
 - f. There is no support for “LOCAL” in the DATE or TIME option of the CREATE PACKAGE statement. If specified, SQLCODE -168 (SQLSTATE 42615) will be generated, indicating an incorrect parameter.
 - g. DB2 for VSE & VM servers do not support cursors declared with the “WITH HOLD” clause. However VM applications may use the “WITH HOLD” clause against other DRDA servers if they support it, except when extended dynamic statements are involved.
5. There is no support for the semantics checking of the Flagger, but the syntax checking of static SQL against the SAA and SQL-89 standards will still be carried out.

DB2 Server for VSE & VM Facility Restrictions

- 1. There is no support for the USERID option of the SQLPREP EXEC.
- 2. There is no support for “USER” in the preprocessing parameter ISOLATION. The system will override USER with CS.
- 3. There is no support for “LOCAL” in the preprocessor parameters DATE and TIME. If specified, SQLCODE -168 (SQLSTATE 42615) will be generated, indicating an incorrect parameter.
- 4. There is no support for the blocking of PUTs. However, the PUT operation will still be supported one row at a time as unblocked inserts.
- 5. The following ISQL commands are not supported when using the DRDA protocol, because they request functions specific to DB2 Server for VM:
 - SET ISOLATION
 - COUNTER
 - SHOW
- 6. The following DBSU commands are not supported when using the DRDA protocol, because they request functions specific to DB2 Server for VM:
 - UNLOAD DBSPACE
 - UNLOAD TABLE
 - UNLOAD PACKAGE
 - RELOAD DBSPACE
 - RELOAD TABLE
 - SET ISOLATION
 - SET UPDATE STATISTICS
 - REBIND PACKAGE
 - REORGANIZE INDEX
- 7. FORTRAN packages and any other packages created by using extended dynamic statements that were created in SQLDS protocol cannot be RELOADED by the DBS Utility in DRDA protocol, or the other way around.

8. Portable packages created under SQL/DS Version 2 Release 2 cannot be RELOADED by the DBS Utility in DRDA protocol.
9. If accounting data is sent from a DRDA application requester to a DB2 for VSE & VM server, only the first 16 bytes of user-defined data⁸ is captured by the server and put into accounting records.

⁸ For example, from DDCS for OS/2 user-defined data can be set by the DFT_ACCOUNT_STR configuration parameter.

Appendix I. Incompatibilities Between Releases

This appendix identifies the incompatibilities that exist between each release of the product and the previous release, going back to Version 1 Release 3.5. There is a separate section in the appendix for each release.

Note on Skipping Releases: If your migration plans call for skipping one or more releases (for example, migrating directly from V2R2 to V3R4), you will still be affected by the incompatibilities introduced by the releases that you are skipping.

Within each section, the incompatibility items are grouped into the following categories:

- SQL and Data
- Application Programming
- System Environment

Definition of an Incompatibility

For the purpose of this appendix, an “incompatibility” is defined to be a part of the product that works differently than it did in the previous release, in such a way that if used in an existing application, it will produce a different result, necessitate a change to the application, or reduce performance. In this definition, “application” can apply to a broad range of things (singly or in combination), such as:

- Application program code
- Specifications for preprocessing application programs
- Interactive SQL queries
- ISQL functions
- DBS Utility functions
- Miscellaneous tools in your operating environment.

This appendix does not describe incompatibilities where certain operations in the current release are less likely to generate an error condition than they did in the previous release, as those changes will only have a positive impact on your applications. (For example, the SUM and AVG column functions no longer overflow as easily because they now use a larger accumulator, and a change to the use of the equal (=) compare predicate with a negative indicator variable now evaluates to UNKNOWN rather than generating an error condition.)

Impact on Existing Applications

Read the appropriate section of this appendix carefully to determine what changes you will need to make to your applications when migrating from one release to the next. You may also want to review the chapter in the *DB2 Server for VM System Administration* manual on migration considerations which discusses some of these incompatibilities in more detail, plus other considerations for each release-to-release migration.

This appendix excludes the numerous changes and enhancements for which no impact on existing applications is anticipated. These are listed in the *Summary of Changes* section (included with each manual) of the appropriate release of the

library. Review that section to see where you could make changes to your existing applications in order to take advantage of some of these enhancements.

V2R1 and V1R3.5 Incompatibilities

SQL and Data

1. Evaluation of HAVING and SELECT Clauses

Prior to V2R1, the HAVING clause was evaluated *after* the SELECT clause. This caused a statement such as the following to fail on a zero divide and generate SQLCODE -802, if a zero part number was encountered:

```
SELECT 200/PARTNO FROM T1
GROUP BY PARTNO HAVING PARTNO > 0
```

In V2R1, the HAVING clause is evaluated *before* the SELECT clause. This means your applications now have the potential of producing different results. In the above example, if a zero part number is encountered, the query does not fail and SQLCODE -802 is not generated.

2. Null Values as a Grouping Criterion

Prior to V2R1, if any row had a null value in one of the columns referenced in a GROUP BY clause, each such row was treated as a separate group.

In V2R1, null values are considered identical for purposes of grouping.

This means that your existing applications may generate fewer rows in the result table than they did in previous releases, since multiple null-value-groups are now consolidated into one group. Any derived column function values will reflect this consolidation (for example, SUM(BONUS)).

3. Negative Decimal Zero Support

Prior to V2R1, the system recognized negative decimal zero as a valid value. However, it did not evaluate positive and negative decimal zero values as equivalent.

In V2R2, any negative decimal zeros found in SQL statements are converted to positive decimal zeros before execution. This means that inserting, updating, or deriving negative decimal zeros, or using them in a comparison, is no longer possible. A utility called SQLZERO is provided which converts all negative decimal zeros in the database to positive decimal zeros.

For a detailed discussion of this topic, see "Elimination of Negative Decimal Zero" in the chapter which discusses migrating from V1R3.5 in the *System Planning and Administration* manual, V2R1 or later.

4. Insertion of Invalid Decimal Values

Prior to V2R1, it was possible to insert invalid decimal data into the database during DATALOAD by specifying string values that were invalid for DECIMAL columns. For example, X'0000' has no sign value.

In V2R1, this is no longer allowed. Doing so will generate SQLCODE -424.

Application Programming

5. Use of ORDER BY Clause with SELECT INTO

Prior to V2R1, the SELECT INTO statement was allowed to contain an ORDER BY clause.

In V2R1, this is no longer allowed. Doing so will generate SQLCODE -524.

6. Scope of Prepared Statements

Prior to V2R1, a prepared statement could sometimes, but not always, be referenced in subsequent logical units of work (LUWs).

In V2R1, this inconsistency is removed. A prepared statement may now only be referenced within the same LUW in which it was prepared.

If your applications contain code that references prepared statements across LUWs, they will have to be restructured accordingly.

7. SQLCODE Returned After a Format 2 INSERT

Prior to V2R1, when a format 2 INSERT (known as “INSERT via subselect” in V2R2 and later releases) returned an empty answer set for insertion, SQLCODE +0 was generated.

In V2R1, SQLCODE +100 is generated instead.

8. Preprocessor Errors Converted to Warnings

Prior to V2R1, a certain set of conditions generated errors during preprocessing.

In V2R1, these conditions now generate warnings, although the associated SQLCODEs are still negative (starting with V3R1, the codes are presented as positive numbers). These conditions and their corresponding SQLCODEs are shown in the table below.

SQLCODE	DESCRIPTION
-134	IMPROPER USE OF THE LONG FIELD COLUMN column.
-135	THE INPUT FOR A LONG FIELD COLUMN IN AN INSERT OR UPDATE MUST BE FROM A HOST VARIABLE OR THE KEYWORD NULL.
-150	THE VIEW CANNOT BE USED TO MODIFY DATA SINCE IT IS BASED ON MORE THAN ONE TABLE.
-151	A COLUMN OF A VIEW CANNOT BE UPDATED SINCE IT IS DERIVED FROM AN EXPRESSION.
-152	A COLUMN OF A VIEW CANNOT BE USED IN A WHERE-CLAUSE SINCE IT IS DERIVED FROM A COLUMN FUNCTION.
-154	VIEW LIMITATIONS DO NOT ALLOW THE USE OF THE FOLLOWING OPERATION: operation
-155	YOU CANNOT PERFORM A JOIN ON A VIEW CONTAINING A GROUP-BY CLAUSE OR A DISTINCT KEYWORD.
-156	RESTRICTIONS APPLY WHEN SELECTING FROM A VIEW CREATED WITH THE DISTINCT OR GROUP BY KEYWORD.
-202	COLUMN column WAS NOT FOUND IN ANY TABLE REFERENCED BY THE COMMAND.
-205	COLUMN column WAS NOT FOUND IN TABLE creator.table.
-401	INCOMPATIBLE DATA TYPES FOUND IN AN EXPRESSION OR COMPARE OPERATION.
-404	A CHARACTER STRING SPECIFIED IN AN INSERT OR UPDATE IS TOO LARGE FOR THE TARGET COLUMN.
-405	THE NUMERIC VALUE, value, IS NOT WITHIN THE RANGE OF THE DATA TYPE.
-407	AN UPDATE OR INSERT OF A NULL VALUE FOR A COLUMN DEFINED AS NOT NULL IS NOT ALLOWED.
-408	AN UPDATE OR INSERT OF A DATA VALUE IS INCOMPATIBLE WITH THE DATA TYPE OF THE ASSOCIATED TARGET COLUMN.
-414	LIKE WAS USED FOR A NUMERIC OR DATE/TIME COLUMN TYPE. IT MUST ONLY BE USED WITH CHAR OR VARCHAR TYPE COLUMNS.
-415	THE DATA TYPES OF CORRESPONDING ITEMS IN THE SELECT-CLAUSES CONNECTED BY A UNION ARE NOT IDENTICAL.
-416	YOU CANNOT SPECIFY A LONG FIELD COLUMN IN THE SELECT-CLAUSE OF A UNION.
-419	THE PRECISION OF THE NUMERATOR AND/OR THE SCALE OF THE DENOMINATOR ARE TOO LARGE FOR DECIMAL DIVISION.
-421	A HEXADECIMAL LITERAL WITH AN ODD LENGTH MAY NOT BE USED WITH A DBCS COLUMN IN A PREDICATE.

V2R2 and V2R1 Incompatibilities

SQL and Data

1. Leading and Trailing zeros in Decimal Constants

Prior to V2R2, leading and trailing zeros of decimal constants were removed by the system when calculating their scale and precision.

In V2R2, if the precision of a decimal constant is greater than 15, leading zeros are removed to bring the precision down to 15. Trailing zeros are not removed.

If your current applications provide output from the result table without any intervening formatting, this change has the potential of altering that output. If formatting is involved, you may have to change the formatting logic to obtain the same output.

Similarly, input to the database by means of INSERT or UPDATE may be affected, if a decimal constant is involved.

2. Use of Host Variables with UNION

Prior to V2R2, two select-lists could be successfully UNION'ed even when they contained corresponding items that were host variables of different data types and different lengths. The statement below is an example of this, where host variables :hw and :fw are halfword fixed binary (15) and fullword fixed binary (31), respectively.

```
SELECT :hw FROM T1
UNION
SELECT :fw FROM T1
```

In V2R2, the above statement is no longer allowed. Issuing it will generate SQLCODE -415.

Note: In V3R1, some restrictions on the use of data types within a UNION are removed, including the above incompatibility.

Application Programming

3. Atomic Operations Against the Database

Prior to V2R2, many types of operational errors (that is, SQL statement errors) against the database caused the system to roll back the entire current logical unit of work (LUW), leaving the application with no control over the status of the LUW.

In V2R2, all operations against the database are now atomic. That is, within an LUW, each operation can succeed or fail separately, with no effect on other operations, provided they do not depend on it. If an operation fails, the application is free to either continue working on the same LUW, or commit the changes made so far, or roll back the LUW. Some system errors, such as deadlocks, still require the entire LUW to be rolled back by the system. Also, atomic operation is not supported for:

- Operations on data located in nonrecoverable storage pools
- Operations on data when running without a log (LOGMODE=N).

As a result of this change, you may want to extend the logic of your LUW processing in your applications.

Note: The next incompatibility item contains a special case of atomic operation.

4. Multiple Row Changes Within an Atomic Operation

Prior to V2R2, if an error occurred during a single operation involving multiple row changes to the database, the database was potentially left in an inconsistent state. (This was one of those operational errors that was not rolled back by the system.) Some of the rows were processed; the rest were not. The only practical way to avoid this inconsistency was to have the application roll back the entire current LUW.

There was one exception to this: in the case of a data definition statement, such as CREATE TABLE, the system itself rolled back the LUW to avoid a partial definition of a table in the catalog. The application had no control over the status of the LUW.

In V2R2, with atomic operation in place, the system automatically undoes that portion of the multiple row operation that was processed prior to the error. This eliminates the potential of an inconsistent database resulting from such an operation, and leaves the application free to control the current LUW as it sees fit.

See “Detailed Notes on V2R2-V2R1 Incompatibilities” on page 509 for an example.

5. Four-Byte Floating-point Data

Prior to V2R2, all floating-point data had to be eight bytes.

In V2R2, it can be four bytes.

This leads to a potential problem in V2R2 for programs that allocate eight bytes when using DESCRIBE on a FLOAT column. When using DESCRIBE, applications should allocate storage based on the SQLLEN of a column (as given in the SQLDA), not the SQLTYPE.

6. Arithmetic and Conversion Errors

Prior to V2R2, an arithmetic or conversion error terminated processing of the statement and generated SQLCODE -802.

In V2R2, these types of errors are tolerated when they involve a host variable that has an indicator variable. In such cases, processing of the SQL statement continues; SQLCODE +802 is generated; a -2 is placed in the indicator variable; and the associated database variable remains unchanged.

If your application is checking for these errors, this could impact its logic. The types of errors that can now be tolerated are:

- Fixed point overflow
- Decimal overflow
- Exponent overflow
- Exponent underflow
- Divide exception.

For more detail, see the *Messages and Codes* manual, V2R2 or later, for SQLCODEs +802 and -802.

7. GRANT Authority for PUBLIC

Prior to V2R2, “WITH GRANT OPTION” in a GRANT statement passed GRANT authority to the user receiving the privilege in question, even when the user was PUBLIC.

In V2R2, when “PUBLIC” and “WITH GRANT OPTION” are used together, the privilege is granted to PUBLIC, but without GRANT authority. In such cases, a warning is given to that effect.

This can impact your current authorization of views or programs, since these objects, which previously could have been grantable (for example, a value of 'G' recorded for a program in catalog table SYSPROGAUTH), will no longer be so (a value of 'Y' now in SYSPROGAUTH) if they depend on PUBLIC access to an object.

For example, if a program contains a static SELECT statement involving table T1, and the owner of the program is dependent on PUBLIC access to T1, then 'Y' is the highest authorization value attainable for that statement — and therefore for the program. This means that the owner is still able to run the program, but not to grant the RUN privilege on it to others. This, in turn, means that when this program is preprocessed under V2R2, users who previously may have had authority to run it (by virtue of receiving RUN authority from the owner) will no longer have that authority.

System Environment

8. Change to Message Numbers

Prior to V2R2, the ARI message numbers were three digits long and were followed by an action indicator. This identification formed a header for each line of the message text, as illustrated below:

```
ARI297A  RESPONSE TO ARCHIVE PROMPT
ARI297A  IS NOT VALID.
```

In V2R2, these message numbers are expanded to four digits to accommodate future expansion of the system. Message numbers existing in the earlier releases now contain a high-order zero. Also, the message header is now only used on the first line of the message. The above example becomes:

```
ARI0297A RESPONSE TO ARCHIVE PROMPT
          IS NOT VALID.
```

This could impact any automated operating system facility that you may be using (for example, the VM Programmable Operator) to scan the message number and text.

Detailed Notes on V2R2-V2R1 Incompatibilities

1. Multiple Row Changes Within an Atomic Operation

In the following example, the operations are contained in one LUW. The second operation involves multiple row changes to the database.

```
DELETE FROM SUPPLIER WHERE SUPPNO = 64
UPDATE INVENTORY SET PARTNO = PARTNO + 1
INSERT INTO QUOTATIONS VALUES (64, 221, .25, 5, 100)
```

The DELETE statement removes a supplier from the SUPPLIER table. The UPDATE statement changes the first two rows of the INVENTORY table, but fails on the third row because the operation would create a duplicate primary key value.⁹

Prior to V2R2, the system would have left the new values in the first two rows of INVENTORY, with the rest of the table unchanged. To avoid this undesirable inconsistency, the application would have had to contain logic to recognize this error and roll back the entire LUW, thus undoing the DELETE.

In V2R2, when this error occurs, the system undoes the UPDATE statement by reversing the changes made to the first two rows. Because neither the DELETE nor the INSERT depends on the success of the UPDATE (these operations are atomic), the application has the following options open to it:

⁹ In V3R2 this error will not occur, because the enforcement of uniqueness is done after all the rows are updated.

- Proceed and perform the INSERT, or
- Commit the successful DELETE, or
- Roll back the LUW to undo the DELETE.

V3R1 and V2R2 Incompatibilities

SQL and Data

1. Table Designation Rules

Prior to V3R1, the following set of ANS/ISO SQL rules for table designation in FROM clauses were not fully enforced:

- Duplicate table or view names in a FROM clause must all have a correlation name assigned to them.
- Correlation names in a FROM clause must be distinct from each other.
- Correlation names in a FROM clause must be distinct from the table or view names in the same clause.

When the application contained ambiguities, such as

```
SELECT A.COL1
FROM A B, B A
```

where COL1 appeared in both table A and table B, the system accepted the statement, employing its own set of rules to resolve the ambiguity. This example represents only one type of ambiguity that could occur.

In V3R1, the ANS/ISO rules are fully enforced. Any violations generate SQLCODE -211 (SQLSTATE 52012).

2. New Reserved Words

Prior to V3R1, the following were not reserved words in SQL and could therefore be used as ordinary identifiers:

- CHAR
- CHARACTER
- DOUBLE
- EXECUTE
- FIELDPROC
- GRAPHIC
- LONG
- PACKAGE.

Similarly, the following were not reserved words for the DBS Utility:

- REORGANIZE
- SCHEMA.

In V3R1, these are reserved words, so an existing application that uses any words in the SQL group above as an ordinary identifier will have to be changed before it is preprocessed, or SQLCODE -105 (SQLSTATE 37501) will be generated. Similarly, the words in the DBS Utility group above can no longer be used in DBS Utility commands as ordinary identifiers.

You can address this incompatibility by changing these ordinary identifiers to use nonreserved words, or you can retain the original names by redefining them as delimited identifiers.

3. Significance of Trailing Blanks

Prior to V3R1, trailing blanks were treated as significant in both object names and VARCHAR and VARGRAPHIC column values.

In V3R1, such trailing blanks are not considered significant.

If your applications must continue to treat trailing blanks as significant, you may have to undertake some redesign. See “Detailed Notes on V3R1-V2R2 Incompatibilities” on page 515 for further discussion and examples.

4. Timestamp at the 24th Hour

Prior to V3R1, a timestamp value in which the hour portion was 24 and the minute, second, or microsecond portion was not zero, was accepted as valid data for insertion or updating.

In V3R1, an attempt to insert or update a column with such a value generates SQLCODE -181 (SQLSTATE 22007). When the hour portion is 24, the other time portions must now be zero.

If you have any of these invalid values in your tables after migrating to V3R1, they will prevent you from doing a DBS Utility unload/reload operation or an INSERT using a subselect. You will have to first correct these values to conform to the rule mentioned above.

Application Programming

5. Invalid Pointers in SQLDA and RDIIN

Prior to V3R1, the system checked for invalid pointers in the SQLDA and RDIIN structures. This checking was extensive, often resulting in poor performance.

In V3R1, in the interest of better performance, this checking has been eliminated. It is up to the application programmer to follow the rules on setting pointers in the SQLDA, as outlined in the chapter “Using Dynamic Statements” in the V3R1 *Application Programming* manual. Pointers in the RDIIN must not be changed by the application. If your application does not satisfy these rules, the results will be unpredictable.

6. Continuation Characters in FORTRAN

Prior to V3R1, the FORTRAN preprocessor ignored any continuation character located in front of an EXEC SQL on the same line, provided it was not part of an IF or ELSE statement — even though such coding was incorrect.

In V3R1, the continuation character is acknowledged and the EXEC SQL is ignored.

7. Missing Comma in COBOL Continuation Lines

Prior to V3R1, if you left out an intended comma from a list of parameters in an SQL statement embedded in a COBOL program (as illustrated below) and did not code a continuation character in the next line, the system would assume a continuation character and misinterpret the parameter list, giving potentially wrong results.

```
SELECT *
FROM T1
WHERE COL1 IN ('AB'    <--- missing comma
               'CD',   <--- no continuation character
               'EF')
```

In V3R1, this error is detected and reported at preprocessor time.

8. DROP PROGRAM Statement Containing Host Variables

Prior to V3R1, the processing of a DROP PROGRAM statement that contained host variables required a specific section in the access module. (In this form of the statement, the name of the owner of the program or the name of the program or both are expressed as host variables.)

Note on New Terminology: As of V3R1, PACKAGE becomes the new reserved word for PROGRAM, the latter remaining as a synonym. Access modules are now referred to as packages. This new terminology is used below.

In V3R1, the host variable form of the DROP PACKAGE statement no longer requires a section in the package. All the information required to execute the statement is sent with the execution-time request. You will be affected if you have this form of the DROP PACKAGE coded in your application programs.

If the programs that use these packages are explicitly reprocessed, they will have to be recompiled (or reassembled) and relinked in order to execute successfully. Otherwise, errors will result, since there will be fewer sections in the new package and this will cause a mismatch between section numbers in the RDIIN structure and the new package.

9. Data Type of String Constants

Prior to V3R1, application programs that assumed that string constants have a data type of VARGRAPHIC because they are used in the context of GRAPHIC and VARGRAPHIC data, were accepted.

In V3R1, such constants are considered to be VARCHAR, and if used in conjunction with GRAPHIC or VARGRAPHIC data will result in an error, such as SQLCODE -171 (SQLSTATE 53015) or SQLCODE -408 (SQLSTATE 53021).

If the host language is COBOL, PL/I, or C, you should use explicitly coded graphic constants. See the section of the V3R1 *SQL Reference* manual that discusses graphic string constants.

10. New Options in CREATE PROGRAM Statement

Prior to V3R1, when the following three options:

```
ISOL({RR|CS|USER})  
DATE({ISO|USA|EUR|JIS|LOCAL})  
TIME({ISO|USA|EUR|JIS|LOCAL})
```

were used in conjunction with an extended dynamic access module, the values for these options were determined when statements referencing the extended dynamic access module were executed. The values were set based on the corresponding preprocessing options of the program containing the extended dynamic statements.

Note on New Terminology: As of V3R1, PACKAGE becomes the new reserved word for PROGRAM, the latter remaining as a synonym of the former. Access modules are now referred to as packages. This new terminology is used below.

In V3R1, these options are added to the CREATE PACKAGE statement, so that they become preprocessing options. This means that their values are stored with the package itself, and are enforced when the sections of the

package are executed. Consequently, your programs may now run at a different isolation level than they did in V2R2.

See “Detailed Notes on V3R1-V2R2 Incompatibilities” on page 515 for examples that illustrate how incompatibilities may arise as a result of this change.

11. Views Created from SELECT *

Prior to V3R1, views created as SELECT * FROM T1 required no special attention when being migrated from release to release, even when columns had been added to table T1 *after* the creation of the view.

In V3R1, a necessary change to the system now requires special attention in the above situation. The first time the system encounters such a view in an application, it attempts to rebuild the view, and fails with SQLCODE -835 (SQLSTATE 56049).

To avoid this failure, drop and recreate the view before running the application on V3R1. Depending on how your application logic is coded, you may have to change that logic in order to handle the extra columns that were added to table T1. The best practice is to avoid the use of SELECT * for view creation, and specify the explicit columns that the application requires.

12. Semicolon Delimiter in SYSVIEW Table

Prior to V3R1, when a view was created through the DBS Utility or by running a preprocessed program, the CREATE VIEW statement was inserted into column VIEWTEXT of catalog table SYSVIEWS with a semicolon delimiter.

In V3R1, this delimiter is no longer included.

If your application has a dependency on the existence of this delimiter in the SYSVIEWS table, you will need to change it accordingly.

13. Replacement of Error Message ARI0565E

Prior to V3R1, error message ARI0565E was issued during preprocessing of FORTRAN programs whenever the input source contained no SQL statements that required creation of a package.

In V3R1, this message is replaced by information message ARI0565I. In addition, related message, ARI0598I, dealing with the status of the package, is modified.

This could impact any automated operating system facility that you may be using (for example, the VM Programmable Operator) to scan the message number and text.

14. Replacement of SQLCODE -150

Prior to V3R1, an attempt to modify data through a view based on more than one table generated SQLCODE -150.

In V3R1, this is replaced with SQLCODE +149 at preprocessor time, and SQLCODE -149 (SQLSTATE 53007) at run time.

15. New Positive SQLCODEs

Prior to V3R1, a number of negative SQLCODEs and associated positive RDSCODEs were returned during preprocessing to indicate a warning situation.

In V3R1, new positive SQLCODEs are returned instead, which correspond identically to the above negative SQLCODEs in code number and (in most

cases) message text and explanation. If the error is not removed, the corresponding negative SQLCODEs will be issued at run time.

See “Detailed Notes on V3R1-V2R2 Incompatibilities” on page 515 for a list of these new positive SQLCODEs.

System Environment

16. Uppercase and Mixed Case in Message Text

Prior to V3R1, all message text was in uppercase for all the languages available in the product except German, which was available only in mixed case.

Note: The uppercase applied to both English language offerings, AMENG and UCENG. It also applied to the English text embedded in the DBCS languages Japanese and Korean (for example, “FORCE,” “SQLEND”).

In V3R1, the message text of three more languages is now changed to mixed case only. These languages are AMENG (the default language setting), Italian, and Spanish. If you are using any of these three languages and you have existing case-sensitive applications that scan for specific message text in uppercase only, you will have to modify them to detect lowercase as well. This could impact any automated operating system facility that you may be using for this purpose (for example, the VM Programmable Operator).

An alternative approach (for English users only) to modifying your applications would be to specify UCENG instead of AMENG, through the SET LANGUAGE command.

17. Authorization for Changing System Catalog Tables

Prior to V3R1, certain portions of the catalog could be updated, deleted, or inserted into, by any user with DBA authority.

In V3R1, the number of columns in the catalog tables for which these changes are allowed is reduced.

This change may affect the authorization of some of your applications. See Appendix E of the V3R1 *SQL Reference* manual for a list of the columns that can now be updated, deleted, or inserted.

18. Modification of Sample Tables and Applications

Prior to V3R1, the sample tables shipped with the product consisted of five Manufacturing tables and four Organizational-project tables. The sample applications shipped with the product used the Manufacturing tables.

In V3R1, the Manufacturing tables are not included, but can be installed optionally. The Organization-project tables are enhanced to provide more guidance on referential integrity and also consistency across the IBM relational database products. The enhancements include:

- Two new tables
- A new column in an existing table
- Renaming of a table
- Modification of a foreign key definition.

The sample applications are now modified to use the enhanced Organization-project tables. They now issue a ROLLBACK instead of a COMMIT, so that they can be rerun without having to first restore the sample database.

If you have any applications that use these tables, such as an online tutorial or a test package for new releases, you will need to upgrade them accordingly.

System Environment (VM Only)

19. Storage of SQLINIT Parameters

Prior to V3R1, the parameters for the SQLINIT EXEC were stored in a bootstrap module. The VM Resource Adapter (VRA) could access this module as long as the module was on one of the accessed minidisks.

In V3R1, these SQLINIT parameters are stored in the LASTING GLOBALV file, which is only accessible by the VRA if this file is on the A-disk. (The CMS GLOBALV manipulation commands require this file to be on the A-disk.)

For example, if you have been executing batch SQL applications by requesting a CMS batch machine to access (as a non-A-disk) the disk where the program and bootstrap reside, you now have to request the CMS batch machine to issue SQLINIT explicitly, so that the proper parameters can be appended to the LASTING GLOBALV file on the batch machine's A-disk.

20. Default Name for Database Machine

Prior to V3R1, the default name of the database machine was SQLDBA.

In V3R1, this default name is now SQLMACH, to distinguish it from SQLDBA, which remains the default name of the database defined on SQLMACH.

This change is only of interest to the system administrator who supplies the name of the database machine during the set up of the SQLDBN file in the XA environment.

Detailed Notes on V3R1-V2R2 Incompatibilities

1. Significance of Trailing Blanks

Prior to V3R1, delimited identifiers "TABLE1" and "TABLE1b" would be considered two different tables, and VARCHAR values 'ABC' and 'ABCbb' two different values, where 'b' represents a blank character.

In V3R1, in the case of the table names, the system would not accept the two tables because they now have identical names. In the case of the VARCHAR values, they are considered equal, except in a LIKE comparison. However, if specified at INSERT or UPDATE time, trailing blanks are included in the varying length string data stored in the database.

If your applications must continue to treat trailing blanks as significant, you may have to undertake some redesign. For example, prior to V3R1, if your table had a VARCHAR column, COLX, containing 'AAAbbb' and you wanted to select all values from COLX that were not equal to 'AAA', the following search condition would satisfy this requirement, because it would return value 'AAAbbb' along with any other values not equal to 'AAA':

```
WHERE COLX <> 'AAA'
```

In V3R1, value 'AAAbbb' does not get returned in the above example. This search condition must be redesigned in order to get the same results as in prior releases. One solution is:

```
WHERE COLX NOT LIKE 'AAA'
```

For more discussion on migration considerations for this item, see "Considerations for VARCHAR and VARGRAPHIC Compare" in the chapter

which discusses migrating from V2R2, in the *System Administration* manual, V3R1 or later.

2. New Options in CREATE PROGRAM Statement

The following examples illustrate the incompatibilities that may arise when you migrate to V3R1.

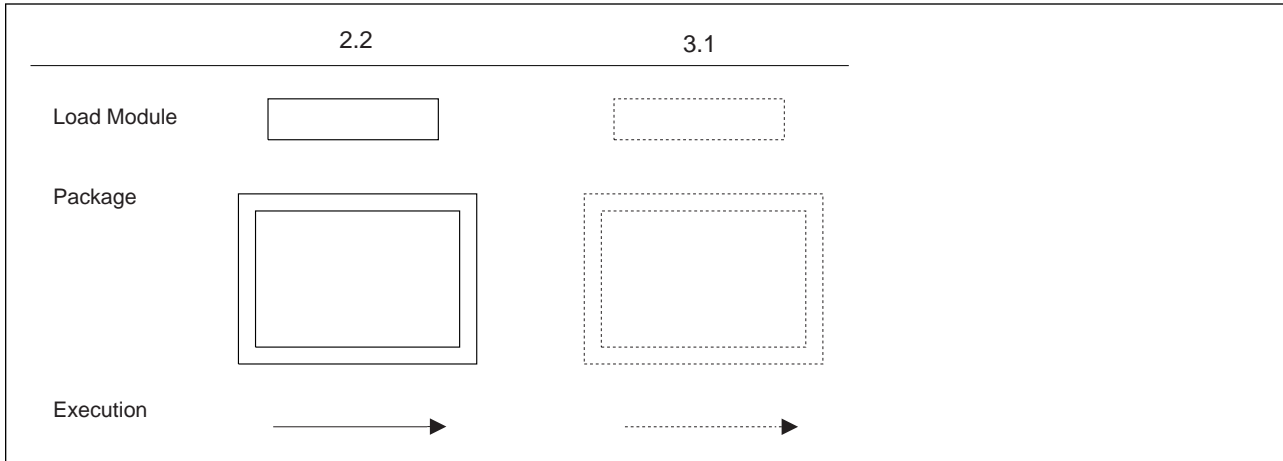


Figure 127. Legend

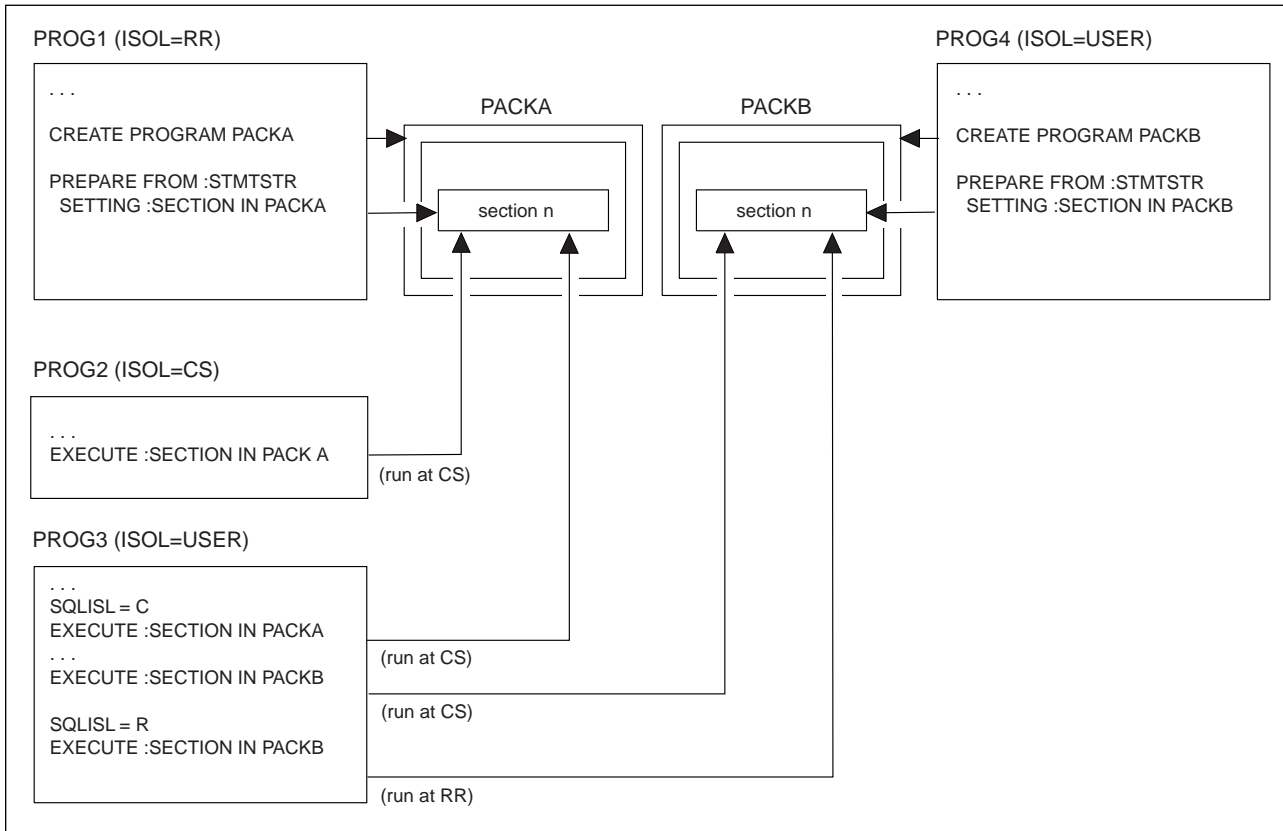


Figure 128. Version 2 Release 2

Figure 128 illustrates how isolation levels are determined for packages created using extended dynamic SQL in V2R2. For example, program PROG1 contains the CREATE PROGRAM statement for package PACKA, and prepares a

section in the package. Program PROG2 subsequently executes the section in PACKA. Since program PROG2 was preprocessed with isolation level cursor stability (CS), the section executes using CS.

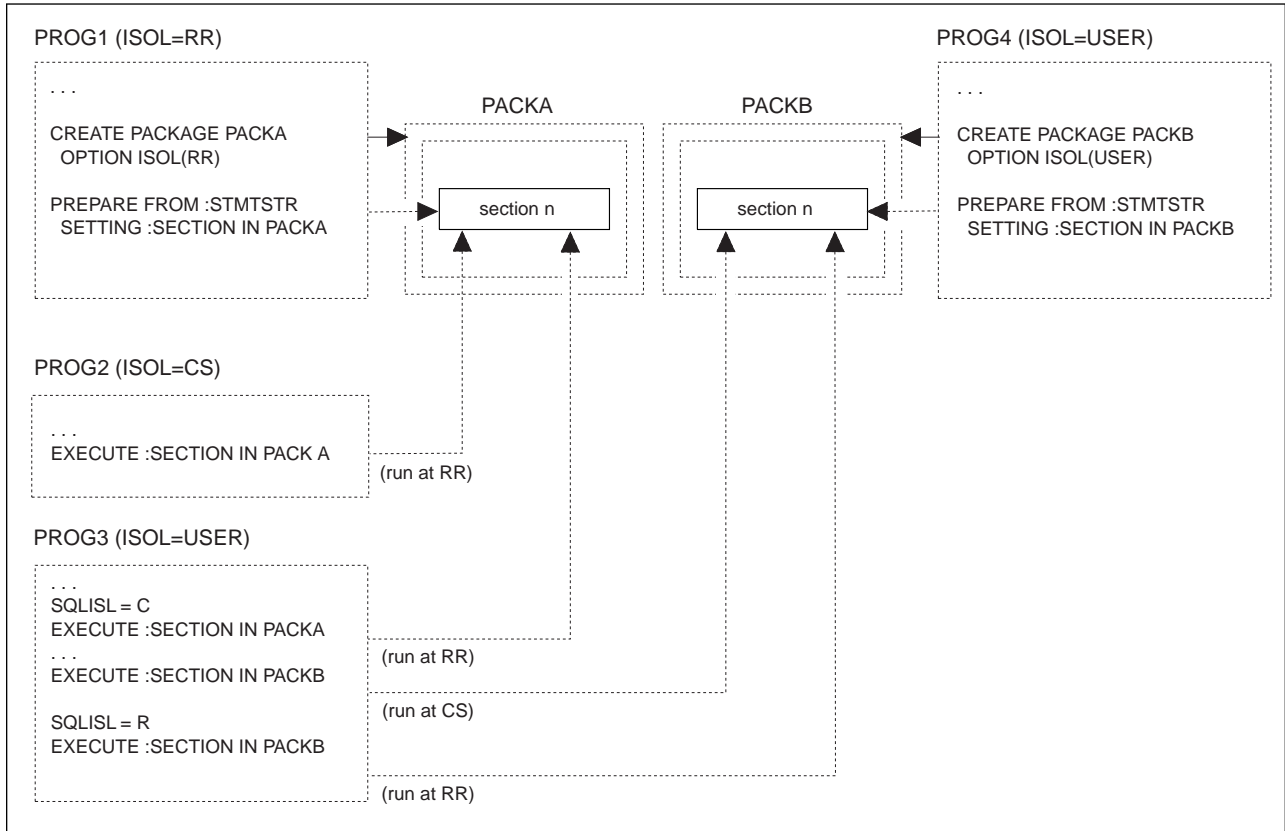


Figure 129. Version 3 Release 1

Figure 129 shows the same scenario in V3R1. In this case, the isolation level RR is specified when the PACKA package is created. When program PROG2 executes a section in PACKA, isolation level RR is used.

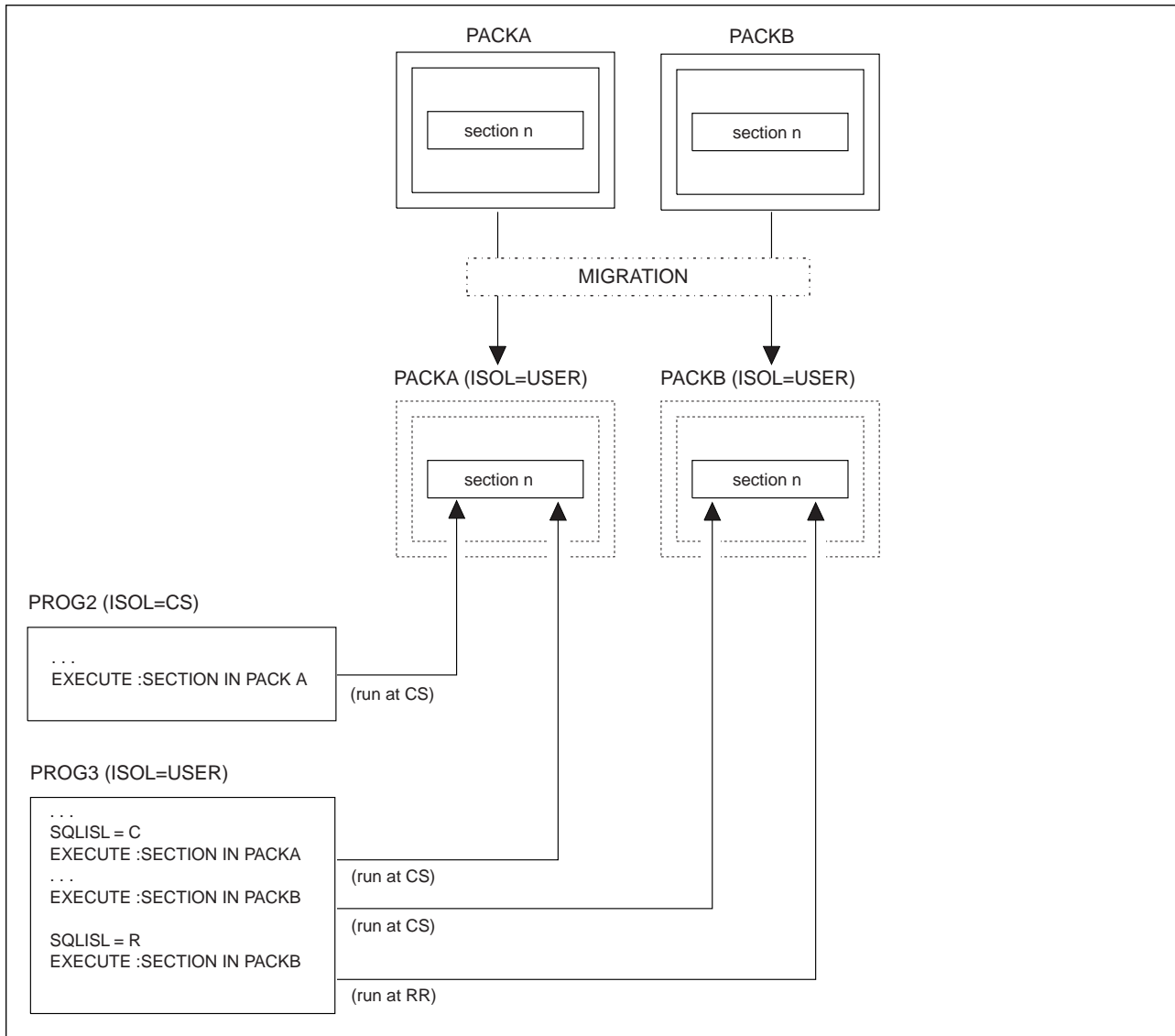


Figure 130. Migration

Figure 130 shows packages being migrated to V3R1. In this case, the isolation level bind option will be automatically set to USER. Applications will notice no change in isolation level handling from previous releases.

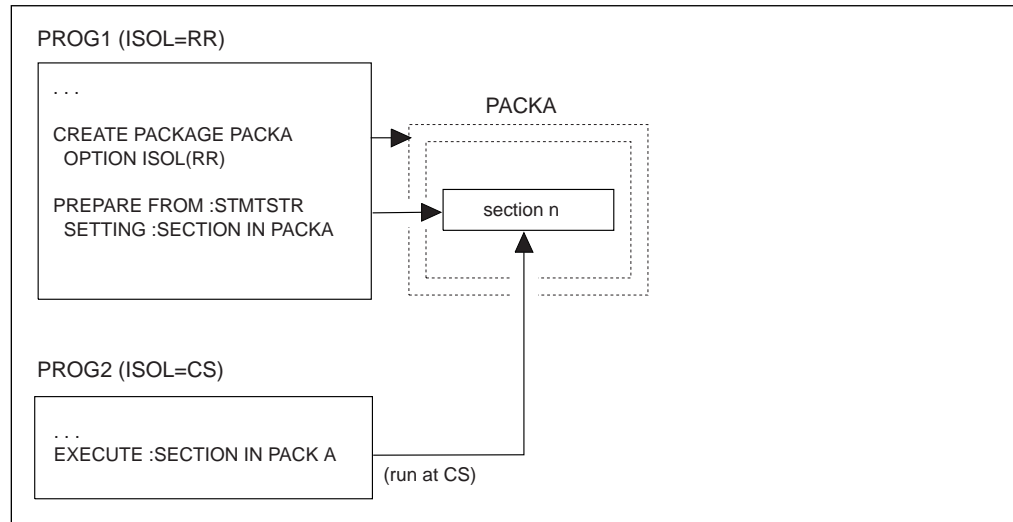


Figure 131. Dropping and Re-creating PACKA Without Reprocessing PROG2

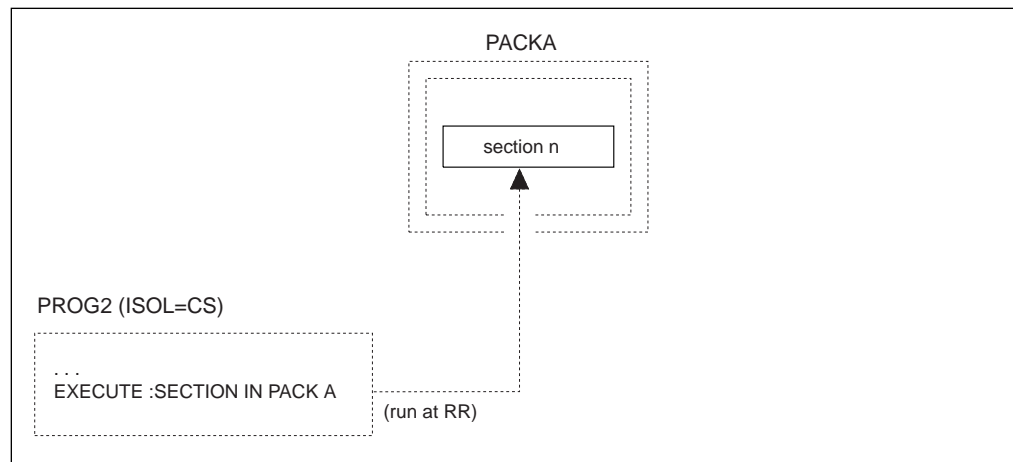


Figure 132. Re-processing PROG2

Figure 131 and Figure 132 show that once an extended dynamic package has been dropped and recreated in V3R1 with an isolation level other than USER, the isolation level bind option will be enforced whenever the executing application has also been preprocessed, assembled, and re-linked under V3R1. If the PACKA package has been dropped and recreated in V3R1, with an isolation level of RR, then:

- If program PROG2 is still pre-V3R1, when the section in PACKA is executed, isolation level CS will be used.
- Otherwise, isolation level RR will be enforced whenever sections in PACKA are executed.

3. New Positive SQLCODEs

These codes are shown in the table below.

SQLCODE	SQLSTATE	DESCRIPTION
+117	01525	The number of data values to be inserted does not equal the number of columns specified or implied.
+134		Improper use of long string.
+135		The input for a long string column in an INSERT statement or UPDATE statement must be from a host variable or be the keyword NULL.
+149		The view cannot be used to modify data because it is based on more than one table.
+151		A column of a view cannot be updated since it is derived from an expression.
+154		View limitations do not allow you to use the following operation: xxxxxx
+202	01533	Column xxxxxx was not found in any table referenced by the statement.
+204	01532	xxxxxx was not found in the system catalog.
+205	01533	Column xxxxxx was not found in table yyyyyy.
+206	01533	The xxxxxx on yyyyyy was not found.
+401		Incompatible data types found in an expression or compare operation.
+404		A character string specified in an INSERT or UPDATE statement is too large for the target column.
+405		The numeric value, xxxxxx, is not within the range of the data type.
+407		Either an UPDATE statement or an INSERT statement with a null value for a column defined as NOT NULL is not allowed, or a null host variable value is not allowed in a SELECT list.
+408		An UPDATE or INSERT of a data value is incompatible with the data type of the associated target column.
+414		The LIKE clause was used for a numeric or date/time column type. LIKE must only be used with character or graphic compatible columns.
+415		The corresponding columns, n, of the operand of a UNION or a UNION ALL do not have comparable column descriptions.
+416		You cannot specify a long string column in the SELECT clause of a UNION.
+419		The precision of the numerator and/or the scale of the denominator are too large for decimal division.
+421		A hexadecimal literal associated with a graphic compatible column in a predicate cannot have an odd length.
+551	01548	User xxxxxx does not have the yyyyyy privilege.
+552	01542	xxxxxx is not authorized to perform this statement.
+668		Table xxxxxx is inactive and you cannot access it.

V3R2 and V3R1 Incompatibilities

SQL and Data

1. Nonexposed Table Names

Prior to V3R2, nonexposed table names (those that have an associated correlation name in the FROM clause) could be referenced within the SQL statement containing such a name.

In V3R2, this is no longer the case. Any application code that makes such a reference will have to be changed to reference the associated correlation name instead. Otherwise, SQLCODE -201 (SQLSTATE 52003) will be generated.

For example, if both tables in the FROM clause

```
FROM TABLE1, TABLE2 A
```

have a column named DESCR, any reference in the query to this column for the second table would have to be written as A.DESCR, not TABLE2.DESCR, because TABLE2 is a nonexposed table name.

2. DISTINCT Column Functions in HAVING Clauses

Prior to V3R2, a DISTINCT column function was allowed in conjunction with a dyadic operator in the predicate of a HAVING clause. (A dyadic operator is an operator having two operands.) For example, the following would be accepted as valid:

```
SELECT JOB, AVG(SALARY), AVG(BONUS)
FROM EMPLOYEE
GROUP BY JOB
HAVING AVG(DISTINCT BONUS) + 50 > 100
```

In V3R2, as part of the product's compliance with SQL-89 in the introduction of unary minus in DISTINCT column functions, this code is no longer allowed. Using it will generate SQLCODE -112 (SQLSTATE 37507).

3. New Reserved Word, SOME

Prior to V3R2, SOME was not a reserved word in SQL and could therefore be used as an ordinary identifier.

In V3R2, SOME is a reserved word that is used in quantified predicates as a synonym for ANY, so any existing applications that use it as an ordinary identifier will have to be changed before they are preprocessed under V3R2. If SOME is used as an ordinary identifier, SQLCODE -105 (SQLSTATE 37501) will be generated.

You can address this incompatibility by changing this ordinary identifier to use a nonreserved word, or you can retain the original name by redefining it as a delimited identifier.

4. Comparing Character Data with Unquoted Numeric Data

Prior to V3R2, applications that compared character data type columns to an unquoted numeric, represented invalid SQL code that was accepted. For example,

```
WHERE C1 = 3
```

where C1 was defined as CHAR(1).

In V3R2, this is no longer accepted. Doing this comparison will generate SQLCODE -401 (SQLSTATE 53018).

5. CHAR Scalar Function with a Timestamp Argument

Prior to V3R2, applications that used a second argument for the CHAR scalar function, when the first argument was a timestamp expression, represented invalid SQL code that was accepted. The second argument was ignored.

In V3R2, this is no longer accepted. Using this argument will generate SQLCODE -171 (SQLSTATE 53015).

6. No Column Name in a Column Function Within a HAVING Clause

Prior to V3R2, applications that used a column function within a HAVING clause with no explicit column name in its argument, represented invalid SQL code that was accepted. For example:

```
HAVING MIN(1) > 30
```

In V3R2, this is no longer accepted. Using this function will generate SQLCODE -111 (SQLSTATE 56001).

7. Even-numbered Precision for Columns

Prior to V3R2, columns that were specified with even-numbered precision were rounded up to the next odd-numbered precision, when creating or altering a table. For example, DECIMAL(6,2) became DECIMAL(7,2) at CREATE time.

Similar rounding up is also performed for arithmetic expressions found inside statements. For example, the expression $99.9999/12*(12+3)$ will become $099.9999/12*(12+3)$ during processing.

In V3R2, this rounding is no longer done. In the above example, any application code that relies on such rounding in order to store seven digits in the column will require a redefinition of the column to DECIMAL(7,2), if the table gets recreated in V3R2. Otherwise, one of the following error conditions (depending on where the mismatch between column and length of the variable occurs) will be generated: SQLCODE -302 (SQLSTATE 22003), SQLCODE -405 (SQLSTATE 53020), or SQLCODE -413 (SQLSTATE 22003).

Arithmetic expression that relies on such rounding to obtain enough precision to accommodate the result of the calculation will need modification. In the above example, the 99.9999 in the expression $99.9999/12*(12+3)$ must be changed to 099.9999 in order to accommodate the result which is 124.99988. Otherwise, SQLCODE -802 (SQLSTATE 22003) will be generated.

Date/time Durations: Date and time durations are specified as DECIMAL(8,0) and DECIMAL(6,0) respectively, but if stored in the database prior to V3R2, they became DECIMAL(9,0) and DECIMAL(7,0) columns. Because of this, V3R2 still accepts the odd-numbered precision for these durations, when they are used as input.

Performance of Assembler Programs: Assembler does not support even-numbered precision. If such table columns are referenced in a predicate containing a comparative host variable in an Assembler program, the latter must be declared with a precision one higher than the column. This leads to inefficient processing. You should consider redefining such table columns to odd-numbered precision to avoid this reduction in performance.

8. Floating-point Ranges

Prior to V3R2, there was a certain range of floating-point values that went beyond the allowable values for the database and if encountered, would generate SQLCODE -405 (SQLSTATE 53020).

In V3R2, because of a necessary change in the checking algorithm for floating-point constants, the following two narrow ranges have been added to the original range and will now also trigger SQLCODE -405 when encountered:

- Approximately +7.2370055773322608E+75 to +7.23700557733622E+75
- Approximately -7.2370055773322608E+75 to -7.23700557733622E+75.

9. Decimal Precision in Internal Sorts

Prior to V3R2, an arithmetic operation involving decimal columns, such as COL1*COL2/100 or SUM(COL3), allowed a precision of up to 15 digits, unless the SQL query specified something less.

In V3R2, with the enhancement of decimal precision, this allowable precision is now expanded to 31 digits. As a result, it is possible for a query that has been migrated from V3R1 to generate SQLCODE -101 (SQLSTATE 54001) with a value of 'ARIXECK' in the SQLERRP field of the SQLCA. This error indicates that the maximum allowable size (255) of an internal sort key has been exceeded. This can only occur if the query is fairly complex and requires an internal sort.

Note: Queries that use internal sorts are typically those that use ORDER BY, UNION, or DISTINCT.

If you experience this error, you can reduce the precision of the arithmetic operations in the select list of your query by applying the DECIMAL scalar function. This, in turn, may reduce the internal sort key to an acceptable length.

10. Quantified Predicates Involving Null Values

Prior to V3R2, null values in quantified predicates (ALL, ANY) were not handled according to the FIPS standard.

In V3R2, the FIPS standard applies. As a result, the truth value of these predicates is different from previous releases for some cases involving null values.

See “Detailed Notes on V3R2-V3R1 Incompatibilities” on page 525 for a discussion on these cases and examples to illustrate the incompatibilities.

Application Programming

11. Negative Indicator Variables in Predicates

Prior to V3R2, the use of negative indicator variables in predicates was limited to the basic equal-to (=) predicate in static, dynamic, and extended dynamic SQL.

In V3R2, the use of negative indicator variables in predicates is extended in some areas and restricted in others. This use is now allowed in all predicates of static SQL and extended dynamic SQL when a descriptor is specified on the PREPARE statement. SQLCODE -309 (SQLSTATE 22512) is generated when a negative indicator variable is used in any predicate within dynamic SQL, or extended dynamic SQL when no descriptor is specified on the PREPARE statement.

12. Declaration of Indicator Variables

Prior to V3R2, existing application programs that used indicator variables declared with a data type other than the equivalent of SMALLINT were accepted.

In V3R2, these programs are no longer accepted. For FORTRAN programs, error message ARI0550E is generated at preprocessing time; for Assembler, C, COBOL, and PL/I programs, SQLCODE -326 (SQLSTATE spaces) is generated at preprocessor time.

13. Incorrect Data Inserted from Variable Length Host Variables

Prior to V3R2, incorrect data could get inserted into the database from a variable length host variable that had a length value greater than the maximum that was defined at preprocessing time.

In V3R2, this is prevented. If it is attempted, SQLCODE -311 (SQLSTATE 22501) will be generated.

14. Incorrect String Representations of Date/time Values

Prior to V3R2, incorrect string representations of date/time values generated errors at preprocessor time.

In V3R2, warning messages are issued instead; then if the string representations are not corrected, they will result in errors at run time.

15. COBOL Host Variable Names

Prior to V3R2, if a COBOL program contained a hyphen (-) in the declaration of a host variable name, this hyphen could be represented as an underscore (_) where the name was used within an SQL statement.

In V3R2, the preprocessor no longer accepts this substitution within the program.

If you have any such substitutions in your COBOL source code, they will have to be converted to hyphens before preprocessing under V3R2.

16. Validation of Host Variables

Prior to Version 3, applications containing any of the SQL statements SELECT, SELECT INTO, UPDATE, INSERT, or DELETE, could be preprocessed from a user machine on one release of the product to a database machine on another release of the same version.

In V3R2, there is a change to the validation of host variables for these statements. As a result, this preprocessing fails when the two releases involved are V3R1 and a later release of Version 3. To circumvent this problem, you must preprocess the application from a user machine at the same release level as the database machine on which you would like the package created before compiling, linking, and executing the application from the user machine.

Application Programming (VM Only)

17. Backslash Continuation Character Embedded in C

Prior to V3R2, the backslash was not recognized as a continuation character in SQL statements embedded in the C language. For example, the following two-line SQL statement ends its first line with a backslash followed by three blanks, each denoted as b:

```
EXEC SQL INSERT INTO T1 VALUES('abcd\bbb  
efg', 2);
```

In these earlier releases, the first value inserted into the table would be:

```
'abcd\bbbefg'
```

In V3R2, the backslash is recognized as a continuation character in SQL statements embedded in the C language. As a result, the first value inserted into the table, using the above example, becomes:

```
'abcdefg'
```

18. SQL Statements Embedded in the C Language

Prior to V3R2, if an SQL statement was followed by a C statement, C comment, or another SQL statement on the same line, this follow-on information was ignored by the preprocessor.

In V3R2, this follow-on information is processed by the preprocessor, with one exception: if the first SQL statement on the line is an INCLUDE statement (other than INCLUDE SQLCA or INCLUDE SQLDA), warning message ARI5406I is issued and the follow-on information is not processed.

19. NUL-terminated Strings in C

Prior to V3R2, if a host variable was declared just one byte too short to hold the NUL byte, the system did not insert the NUL byte into the host variable and no truncation of the data occurred. SQLWARN1 in SQLCA was set to 'N' in such cases.

In V3R2, the database manager interprets a character string in C that has a length greater than 1 as a NUL-terminated string. It puts a NUL byte at the end of the string, even though this may cause truncation. This applies when the data type from the database is either character or date/time. In the case of the host variable being declared just one byte short of the actual length of the data, SQLWARN1 is no longer set to N. It is treated the same as other truncation cases: SQLWARN1 is set to W, and the last byte of the declared length of the host variable becomes a NUL byte.

20. SQL Case-Sensitive Names in C

Prior to V3R2, statement names and cursor names in SQL statements embedded in C were case-sensitive (that is, a letter coded in lowercase would not be equal to the same letter coded in uppercase).

In V3R2, these names are folded into uppercase.

This will require a change to your application code, if you have used multiple statements or cursor names that differ only by their case sensitivity.

Detailed Notes on V3R2-V3R1 Incompatibilities

1. Quantified Predicates Involving Null Values

Those cases for which your applications will give different results than they did in earlier releases can be divided into three types, as described below. The accompanying examples are based on these two tables, where the question mark represents a null value:

Table T1:	C1	C2	Table T2:	C3
	--	--		--
	?	1		?
	2	2		2

Recalling that a quantified predicate involves the structure

<expression> <quantifier> <subquery>

the three types can be described as follows:

a. Prior to V3R2, when

- The value of the expression is NULL, and
- The subselect returns an empty set,

the truth value of the quantified predicate was UNKNOWN.

In V3R2, the truth value is TRUE if the quantifier is ALL, and FALSE if the quantifier is ANY.

In the example below, the second row of T1 is returned by any release of the database manager, but the first row of T1 is only returned by V3R2.

```
SELECT * FROM T1
WHERE C1 > ALL (SELECT C3 FROM T2 WHERE C3 > 2)
```

b. Prior to V3R2, when

- The quantifier is ALL, and
- The subselect returns at least one NULL, and
- There are no values in the result of the subselect for which the *implied* predicate (the predicate applied to just one value in the result) is FALSE;

or when

- The quantifier is ANY, and
- The subselect returns at least one NULL, and
- There are no values in the result of the subselect for which the *implied* predicate (the predicate applied to just one value in the result) is TRUE,

the truth value of the quantified predicate was FALSE, except when the expression was NULL.

In V3R2, the truth value is UNKNOWN.

Note: This change will only affect the results of queries in which a NOT has been applied to the quantified predicate in the situations described above. When a NOT is applied, the truth value is TRUE for prior releases, but is UNKNOWN for V3R2.

In the example below, both rows of T1 are returned by previous releases, but only the first row of T1 is returned by V3R2:

```
SELECT * FROM T1
WHERE NOT C2 = ALL (SELECT C3 FROM T2)
```

See the following references for performance implications of queries similar to those shown in the above examples:

- Chapter 2 of the V3R2 *Database Administration* manual for a discussion on nulls in quantified predicates where null columns are allowed, under “Creating Tables.”
- Chapter 5 of the V3R2 *Diagnosis Guide and Reference* manual for a discussion on inefficient search where nullable expressions are involved, under “Analysis of Performance Problems.”

c. Prior to V3R2, when:

- The expression contains an arithmetic expression, scalar function or column function
- The quantifier is ALL
- The subselect returns at least one NULL, and
- There are no values in the result of the subselect for which the *implied* predicate (the predicate applied to just one value in the result) is FALSE,

the truth value of the quantified predicate was TRUE, except when the expression was NULL.

In V3R2, the truth value is UNKNOWN.

In the example below, the second row of T1 is not returned by any release of the database manager. However, the first row of T1 is returned by previous releases, but not by V3R2.

```
SELECT * FROM T1
WHERE C2 + 1 = ALL (SELECT C3 FROM T2)
```

Comparison of types (b) and (c): Type (c) is really a subset of the more general case outlined in type (b), by virtue of its extra condition about the expression. However, type (c) is included separately here, because it represents an **exception** to the more general case. The exception lies in the fact that these two types generated different results for the truth value prior to V3R2. In V3R2, however, this exception disappears, because their results are now the same (truth value = UNKNOWN).

V3R3 and V3R2 Incompatibilities (VM Only)

Note: This section does not include the restrictions on the use of DRDA protocol, as that topic is covered in the appendix describing DRDA considerations.

SQL and Data

1. New Reserved Word, CONCAT

Prior to V3R3, CONCAT was not a reserved word in SQL and could therefore be used as an ordinary identifier.

In V3R3, CONCAT is a reserved word, and can be used as an alternative to the concatenation operator (||). Any existing applications that use it as an ordinary identifier will have to be changed before they are preprocessed under V3R3; otherwise SQLCODE -105 (SQLSTATE 37501) will be generated.

You can address this incompatibility by changing this ordinary identifier to use a nonreserved word, or you can retain the original name by redefining it as a delimited identifier.

2. REVOKE UPDATE

Prior to V3R3, the REVOKE statement for the UPDATE privilege ignored any column names that might be present as parameters of the UPDATE option — even though such coding was invalid. (This statement is only done on a table basis, never a column basis.)

In V3R3, such parameters are not allowed. If they are used, SQLCODE -105 (SQLSTATE 37501) will be generated.

3. Numeric Data in Character Strings

Prior to V3R3, columns with a data type of CHAR or VARCHAR accepted numeric data, including FLOAT, on insert or update. For example, the following statements did not create an error:

```
CREATE TABLE T1 (COL CHAR(8))
CREATE TABLE T2 (COL VARCHAR(8))

INSERT INTO T1 (123)
INSERT INTO T2 (123)
INSERT INTO T1 (1E1)
INSERT INTO T2 (1E1)

UPDATE T1 SET COL = 123
UPDATE T2 SET COL = 123
UPDATE T1 SET COL = 1E1
UPDATE T2 SET COL = 1E1
```

In V3R3, these inserts and updates now generate SQLCODE -408 (SQLSTATE 53021).

If you want to use the value 123, you must now use it as a character literal ('123'). Float literals are no longer allowed for character columns.

4. Invalid String Representation of Datetime

Prior to V3R3, when a predicate was being evaluated that contained an operand that was one of the special registers CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP, and one of the other operands was a character column of the correct length but containing a value that was not a valid string representation of a datetime, the application ran successfully. Any row containing such an invalid value was returned if it met the search condition. For example, all invalid date values in column, ORDERDATE, were returned for the following condition:

```
WHERE CURRENT DATE <> ORDERDATE
```

In V3R3, SQLCODE -180 (SQLSTATE 22007) is generated under the above condition.

5. Internally Generated Table Names

Prior to V3R3, the system internally built a composite table name that included the name of the relational database, based on a certain maximum length.

In V3R3, this length is slightly increased, and the internal process is now common to the SQL/DS and DRDA protocols. As a result, there is a very small probability that some of your SQL statements could exceed an internal limitation of the system and generate an SQLCODE -101 (SQLSTATE 54001).

The more table names you have in a statement, the greater the probability of this occurring. If you experience this error, one possible solution would be to break the statement down into two separate statements.

Application Programming

6. Setting of SQLN Field

Prior to V3R3, if field SQLD in the SQLDA area held a greater value than the SQLN field after a DESCRIBE, the system set SQLN to zero.

In V3R3, the value of SQLN is not changed.

If your application tests SQLN for zero to verify successful completion of the DESCRIBE, the logic will have to be revised to test for SQLD > SQLN.

7. C NUL-Terminated Strings - Variable Length

Prior to V3R3, a C input string with a length greater than 1 was treated as a fixed length character host variable. It was not mandatory to have a NUL present in it except when the input host variable length was 255, in which case SQLCODE -426 (SQLSTATE 22523) was generated.

In V3R3, a C input string is no longer treated as fixed length. A NUL must be present on all C NUL-terminated input strings except those with a length of 1; otherwise SQLCODE -302 (SQLSTATE 22001) is generated. SQLCODE -426 (SQLSTATE 22523) is no longer generated.

8. C NUL-Terminated Strings - NUL Byte

Prior to V3R3, the NUL byte in a C NUL-terminated string was treated as a blank.

In V3R3, it is treated as a string terminator.

9. C NUL-Terminated Strings - Trailing Blanks

Prior to V3R3, any trailing blanks in a C NUL-terminated string were removed when using the string to update or insert a VARCHAR column or to compare to a VARCHAR column.

In V3R3, these blanks will no longer be removed.

10. C NUL-Terminated Strings - Length

Prior to V3R3, the scalar function, LENGTH, with a C NUL-terminated string as its argument, returned the defined length.

In V3R3, this function now returns the length according to the position of the NUL terminator. (This length excludes the terminator itself.)

11. SQL Statement String

Prior to V3R3, an SQL statement string could end with a statement terminator, when used in conjunction with EXECUTE IMMEDIATE, PREPARE, or Extended PREPARE. An example of such a statement is

```
DROP TABLE T1;
```

which has a trailing semicolon. This was allowed in application programs, even though such coding was invalid. It was also allowed in ISQL and QMF*, since those facilities also use the above three statements to process interactively issued statements.

In V3R3, this statement terminator is not allowed. If it is used, SQLCODE -104 (SQLSTATE 37501) will be generated.

If you have been using such a terminator for the CREATE VIEW statement, your use of catalog table SYSVIEWS could be affected, as described in the *DB2 Server for VSE & VM SQL Reference* manual.

12. Preprocessing of Extended Dynamic Statements

Prior to V3R3, a cursor-variable with a defined length greater than 18 was accepted by the preprocessor, even though such variables should only be defined with a length of 18.

In V3R3, the preprocessor traps this condition and generates SQLCODE -324 (SQLSTATE spaces). You will have to change any applications that use these invalid cursor-variable lengths in your extended dynamic statements.

13. Data Type of Hexadecimal Constants

Prior to V3R3, application programs that assumed that hexadecimal constants have a data type of VARGRAPHIC, because they are used in the context of GRAPHIC and VARGRAPHIC data, were accepted.

In V3R3, such constants are considered to be VARCHAR. If used in conjunction with GRAPHIC or VARGRAPHIC data, they will cause a number of specific SQLCODEs and corresponding SQLSTATEs, dependent on individual cases.

This also means that SQLCODE -421 (SQLSTATE 53055), dealing with hexadecimal literals of odd length, is no longer generated.

14. Non-updatable View

Prior to V3R3, a user with DBA authority who tried to update a view that was not updatable got an appropriate error, such as SQLCODE -154 (SQLSTATE 56009). A user without DBA authority, however, got an authorization error, SQLCODE -551 (SQLSTATE 59001).

In V3R3, the latter user receives the same error message as the DBA user, instead of the authorization message.

15. SYSTEM Table Missing from the System Catalog

Prior to V3R3, if you tried to INSERT, DELETE, or UPDATE a table or view created by 'SYSTEM', but which was not in the system catalog, SQLCODE -823 (SQLSTATE 53032) was generated, indicating that you lacked proper authorization.

In V3R3, SQLCODE -204 (SQLCODE 52004) is generated instead, indicating that the object could not be found in the system catalog.

16. Folding of Lowercase in PREP and DBSU

Prior to V3R3, folding of lowercase into uppercase in PREP and the DBS Utility was done by adding X'40' to the hexadecimal representation of the lowercase character. Sometimes this resulted in characters being folded incorrectly (for example, in the Katakana character set).

In V3R3, this is done using the 370 built-in Assembler instruction TRANSLATE and the user-specified character translation table, in order to be consistent with how the application server handles this operation. One exception to this is when the DBS Utility processes SCHEMA input files. Folding is no longer done on these files; this makes it consistent with the DBS Utility control file, which only allows uppercase input.

If your applications have built-in dependencies on the previous folding scheme, you could get different results. For example, a Katakana user may have a character in his or her coding scheme that has a hexadecimal value that appears to the database manager as one of the 26 lowercase English letters.

Instead of being folded to uppercase English, the Katakana character will now be folded according to the Katakana character translation table.

If you have lowercase in your DBS Utility SCHEMA input file, you will have to change it to uppercase.

17. Loading Audit Trace

Prior to V3R3, the *Database Administration* manual contained sample table definition and DATALOAD parameters for creating a security audit table and loading trace records into it.

In V3R3, the position of the columns within the table are changed and a new column, EXTLUWID, added. If you have been loading audit trace data using this table definition and a DATALOAD job, you will need to change the DATALOAD job, as documented in the V3R3 *Database Administration* manual. If you also want to make use of the new EXTLUWID column, you will need to recreate the table as well.

18. Switching Databases without Connect Authority

Prior to V3R3, if you attempted to switch databases and did not have connect authority for the new database, SQLCODE -561 (SQLSTATE 42505) was generated as a warning situation. It was possible to continue processing on the original database with a non-CONNECT statement.

In V3R3, this situation is treated as a severe error, SQLWARN0 and SQLWARN6 are set to 'S', and any subsequent non-CONNECT statement results in termination of the application. Only a CONNECT statement is accepted.

19. SQLCODE Generated by Operator FORCE Command

Prior to V3R3, either SQLCODE -933 (SQLSTATE 57027) or SQLCODE -948 (SQLSTATE 57027) was returned to the application, when the operator issued a FORCE command to roll back the current logical unit of work.

In V3R3, only SQLCODE -933 (SQLSTATE 57027) is returned.

20. SQLSTATE Changes

Prior to V3R3, certain SQLCODEs had associated SQLSTATEs that did not conform to the SAA standards.

In V3R3, these SQLSTATEs are replaced with ones that do conform. See “Detailed Notes on V3R3-V3R2 Incompatibilities” on page 533 for a list of these codes, along with their old and new SQLSTATEs.

System Environment

21. The Use of DBCS Characters with the CHARNAME Setting

Prior to V3R3, you could use graphic or mixed constants, the VARGRAPHIC scalar function, or you could define columns as GRAPHIC or FOR MIXED DATA, independent of the CHARNAME setting on the application server. Furthermore, you could use graphic or mixed constants, independent of the CHARNAME setting on the application requester.

In V3R3, the above usages result in error conditions such as SQLCODE -640 (SQLSTATE 56031) and SQLCODE -332 (SQLSTATE 57017), if the corresponding CHARNAME does not define a character set with mixed CCSID (that is, if CCSIDMIXED = 0).

22. Setting of CHARNAME

Prior to V3R3, if no CHARNAME was specified, SQLSTART defaulted to CHARNAME = ENGLISH.

In V3R3, it defaults to the CHARNAME used on the previous invocation. If the CHARNAME setting does not define a character set with mixed CCSID (that is, if CCSIDMIXED = 0), then the default character subtype (CHARSUB) will be forced to a value of SBCS.

See the *V3R3 System Administration* manual for the initial default CHARNAME value after installation or migration.

23. Addressing Mode 31-Bit

Prior to V3R3, application programs running in single user mode in a VM environment of XA, ESA 1.0 ESA, or ESA 1.1 ESA, as well as any user exits (accounting, datetime, or field procedures) executed in these environments on the database machine, whether single or multiple user mode, only ran in 24-bit addressing mode.

In V3R3, if the database manager is running in 31-bit addressing mode (AMODE 31) on the database machine, the above application programs and user exits will also run in this mode.

If you have application programs or user exits that fit into this category, you must do one of the following:

- Ensure that they can accommodate 31-bit addressing mode
- Operate the database machine in 370 mode
- Set the AMODE SQLSTART parameter to 24 to force the database manager to run in 24-bit addressing mode.

For information on converting your applications to accommodate 31-bit addressing mode, see the *VM/XA* Application Conversion Guide*. For more information on single user mode and user exits, see the *System Administration* manual.

24. Section Size in a Package

Prior to V3R3, during the preprocessing of a program, the system allocated a section size for each statement in the package.

In V3R3, due to other design changes, it is necessary to increase the size of these sections for SELECT statements. As a result, when an existing package is subjected to a dynamic reparation, it may cause the dbspace to become full, generating SQLCODE -946 (SQLSTATE 57025).

If this occurs in your installation, you will have to explicitly prepare the program with the SQLPREP EXEC, making sure that you have a dbspace that can accommodate the revised package.

Also, the larger sections increase the amount of virtual storage required to run the package. For example, if you have many dynamic SELECT statements in a logical unit of work, they will use up more storage than in the previous release.

25. Three-Part Object Names

Prior to V3R3, an object that was created on a database named (for example) DBX could be successfully referenced later by an application, even though the name for that database had been changed (to, say, DBY). All you had to do

was use the revised name, DBY, when you established the database for the application by means of the SQLINIT EXEC.

In V3R3, the system maintains the name of the database that was used at the time of the object's creation (DBX in this example), as the first part of the object name, thereby making it a three-part name. If you now establish the database for the application under a different name (for example, DBY) the system uses that name as the new qualifier when you try to reference the object. This results in a mismatch of object names and causes SQLCODE -114 (SQLSTATE 56061) to be generated.

This problem can be avoided by simply not changing the names of your databases.

26. Special Characters for CONCAT Operation and Not Equal Condition

Prior to V3R3, the class of the hexadecimal values in the table below was 0.

CHARNAME	Hexadecimal Values
ENGLISH	X'5A', X'B0'
FRENCH	X'BA', X'BB'
GERMAN	X'BA', X'BB'
ITALIAN	X'BA', X'BB'
KATAKANA	X'5A', X'B0'
SPANISH	X'BA', X'BB'

In V3R3, the class of these hexadecimal characters is changed to 6. This is reflected in the CHARCLASS column values of the SYSTEM.SYSCHARSETS catalog table. This change provides additional special characters that can be used to depict the CONCAT operation and the not equal condition in SQL syntax. This, in turn, provides greater flexibility in the use of these two SQL facilities between application requesters and servers that are assigned different CHARNAMES.

This could affect your applications, if they are dependent on previous reclassifications of any of the above characters from class 0 to class 3, for use in ordinary identifiers. For example, if you had reclassified the explanation mark (!) so that DANGER! could be used as an ordinary identifier, this will no longer work because the explanation mark is one of the characters that is now assigned to class 6.

See the *DB2 Server for VM System Administration* manual for details on these classifications.

Detailed Notes on V3R3-V3R2 Incompatibilities

1. SQLSTATE Changes

These changes are shown in the following table.

SQLCODE	Old SQLSTATE	New SQLSTATE	DESCRIPTION
-131	53004	22019	Either the LIKE predicate has an invalid escape character, or the string pattern contains an invalid occurrence of the escape character.
-551	59001	42501	User wwwwww does not have the xxxxxx privilege to perform yyyyyy on zzzzzz.
-552	59002	42502	xxxxxx is not authorized to yyyyyy.
-554	59002	42502	You cannot grant a privilege to yourself.
-555	59002	42502	You cannot revoke an authority or a privilege from yourself.
-556	59002	42502	An attempt to revoke a privilege from xxxxxx was denied. Either xxxxxx does not have this privilege, or yyyyyy does not have this authority to revoke this privilege.
-556	59004	42504	An attempt to revoke a privilege from xxxxxx was denied. Either xxxxxx does not have this privilege, or yyyyyy does not have this authority to revoke this privilege.
-558	59004	42504	You cannot revoke an authority from xxxxxx because xxxxxx has DBA authority.
-560	59005	42505	A CONNECT statement contains an incorrect password for xxxxxx.
-561	59005	42505	User xxxxxx does not have CONNECT authority.
-566	59001	42501	User ID xxxxxx does not have authorization to modify package yyyyyy.
-606	59002	42502	The COMMENT ON or LABEL on statement failed because the specified table or column is not owned by xxxxxx.
-610	59002	42502	The statement failed because a user without DBA authority attempted to create a table in a DBSPACE owner by another user or by the system.
-708	59002	42502	You cannot ALTER, LOCK, or DROP a PUBLIC DBSPACE because you do not have DBA authority.
-713	37515	53015	Incorrect isolation level value xxxxxx specified. Only values C or R may be used.
-801	22004	22003	Exception error xxxxxx occurred during yyyyyy operation on zzzzzz data.
-802	22004	22003	Exception error xxxxxx occurred during yyyyyy operation on zzzzzz data, position nnnnnn. psw1 psw2.
-815	59005	42502	CONNECT denied by accounting user exit routine.
-30053	59006	42506	Owner xxxxxx authorization failed.

V3R4 and V3R3 Incompatibilities (VM Only)

Note: This section does not include the restrictions on the use of DRDA protocol, as that topic is covered in the appendix describing DRDA considerations.

SQL and Data

1. Enhanced EXPLAIN Tables

Prior to V3R4, the tables used by the EXPLAIN statement had some major differences from the corresponding tables in the DB2* product.

In V3R4, these differences are minimized to enhance the EXPLAIN functions and make them more compatible with those in the DB2 product. As a result, there are significant changes to the design of these tables, and the EXPLAIN statement no longer works on the old tables. These changes include new columns dispersed among old ones, the loss of one column, a column data type change, and a column length change.

See the *DB2 Server for VSE & VM SQL Reference* manual for the new design of these tables.

If you have used the EXPLAIN tables in prior releases, you will have to recreate the revised tables before using the EXPLAIN statement in V3R4. To assist you in this task, a DBSU job file containing the necessary create statements is now included as a MACRO file (called ARISEXP) with the product.

Similarly, if you have applications which depend upon the design of the old EXPLAIN tables, you will need to modify these applications to reflect the new design.

Application Programming

2. Reason Codes for Incorrect Host Variable Declarations

Prior to V3R4, a large number of SQLERRD1 codes were associated with SQLCODE -314 (SQLSTATE spaces) at preprocessor time for invalid host variables.

In V3R4, with the introduction of host structures and the associated parsing of declaration statements by the preprocessor, the values of some of these SQLERRD1 codes have changed.

If your application has dependencies on specific SQLERRD1 values, you should look for these changes in the *DB2 Server for VM Messages and Codes* manual and modify your application accordingly.

3. Structured Declarations in COBOL and C

Prior to V3R4, there were a number of error situations for structure declarations in the SQL DECLARE SECTION that were not checked by the COBOL and C preprocessors.

In V3R4, these situations are subjected to validation checks, resulting in the following potential errors, which must be corrected before compilation:

SQLCODE	SQLSTATE	Condition
-107	54003	Host variable name too long
-307	spaces	Duplicate host variable names
-314	spaces	Syntax and semantic errors in a host variable

4. Qualified Field Names in RPG

Prior to V3R4, it was not necessary to qualify the name of a field or subfield in an SQL statement, when that field or subfield name had been duplicated in more than one data structure.

In V3R4, you must qualify these names as follows:

- file-name.field-name
- DS-name.subfield-name

The preprocessor needs this information in order to interpret the reference. If the qualifier is missing, a preprocessor ARI5370E message is generated.

5. Use of Structures in RPG as Host Variables

Prior to V3R4, when the database manager referenced an RPG structure as a host variable, one of two things happened:

- If the structure contained one or more subfields, the database manager accepted the reference. The structure was interpreted as a single character field with a length equal to the length of the total structure.
- If the structure contained no subfields, the database manager rejected the reference, generating an error message.

In V3R4:

- If the structure contains one or more subfields, the reference to it is now interpreted as a reference to each subfield in the structure, giving unpredictable results and potential errors at execution time.
- If the structure contains no subfields, the reference to it is now interpreted as a reference to a fixed length character string with a length equal to the length of the data structure.

Note: Individual subfields within a structure can still be directly referenced as valid host variables. There is no change to this.

If your application references RPG structures as host variables, you will have to change either the declaration section or the SQL statements affected.

6. Application Programs in an Unconnected State

Prior to V3R4, if an application program was connectable but in an unconnected state as a result of a severe error (SQLWARN6 = S) and issued a non-connect SQL statement, the database manager initiated an abend of the application.

In V3R4, SQLCODE -900 (SQLSTATE 51018) is generated and the abend does not occur. If your application is dependent on the abend scenario in this situation, you will have to change it. Otherwise, it may enter an infinite loop.

7. Use of Host Variables in CONNECT Statement

Prior to V3R4, if you used a host variable for the userid or password in a CONNECT statement and the data type of that variable did not satisfy one of the conditions listed below, an error was generated at run time:

- C programs: C-NUL string of length 9
- Assembler, COBOL, or PL/I programs: fixed length character string of length 8.

In V3R4, these conditions are checked by the preprocessor. If they fail the check, SQLCODE -324 (SQLSTATE spaces) is generated.

8. Data Types of Parameter Markers in Predicates

Prior to V3R4, the resolution of data types for a parameter marker was dependent on the highest order of the data types of all the operands to the left

of the parameter marker. Highest order, in the case of numeric operands, implies FLOAT > DECIMAL > INTEGER > SMALLINT.

In V3R4, this resolution process is changed to become more consistent with the DB2 product. If there is an operand expressed as a column name in a BETWEEN predicate, the data type of any parameter marker is resolved as that of the leftmost such operand. Otherwise, the data type of the parameter marker is resolved as that of the leftmost operand that is not a parameter marker — whether in a BETWEEN predicate or an IN predicate.

This could cause a different result from previous releases for predicates that can have more than two operands (namely BETWEEN and IN), but only if your application assigns parameter marker values that are inappropriate for your data.

See “Detailed Notes on V3R4-V3R3 Incompatibilities” on page 538 for some examples and further discussion.

9. Bad Input Records in DATALOAD

Prior to V3R4, a bad input record would terminate DATALOAD command processing on multiple tables when the DBS Utility was running in multiple user mode — whether or not it was preprocessed with the NOBLOCK option. An insert error would be indicated with one of the following codes, followed by message ARI0862E:

SQLCODE	SQLSTATE
-405	53020
-424	22502
-530	23503
-802	22003, 22012, or 22502
-803	23505

In V3R4, such command processing is no longer terminated, if the DBS Utility is preprocessed with the NOBLOCK option. The error indications are still generated, but the processing skips over the bad record and continues.

If you have a dependency in your application on this termination approach prior to V3R4, you may want to address this change in the case of the NOBLOCK option.

10. Index Dependency of a Package

Prior to V3R4, when a SELECT DISTINCT was applied to a single column that had a unique index, the system assumed uniqueness within the column, rather than applying a sort. However, this kind of index dependency was not recorded in the package.

In V3R4, this technique now records the index dependency in the package (for system integrity), even though the index is not actually used to access the table. In addition, the technique is extended to column functions that use DISTINCT — for example, SELECT COUNT(DISTINCT(COL4)), where COL4 has a unique index.

If the index is dropped, the package will now be marked as invalid, causing a dynamic reprep. After the reprep, the application will take longer to execute, because a sort will be needed to process DISTINCT correctly.

System Environment

11. Invocation of TRACE for Storage

Prior to V3R4, if you specified level 2 trace for the STAT or PA component of the TRACDBSS or TRACRDS parameter, respectively, when starting the database manager, you received the Working Storage Manager tracing.

In V3R4, you can use the same specifications, but the Working Storage manager tracing is no longer part of the output.

In order to get this part, you must now use the TRACSTG parameter, or select the STG component when using the TRACE operator command. The format from this trace is different.

12. DBCS Data Conversion Errors

Prior to V3R4, if there was a loading error in a DBCS data conversion routine, SQLCODE -332 (SQLSTATE 57017) was generated with reason code 9. If there was a dropping error in a DBCS data conversion routine, SQLCODE -901 (SQLSTATE 58004) or SQLCODE -30020 (SQLSTATE 58009) was generated.

In V3R4, the above codes are replaced with SQLCODE -674 (SQLSTATE 57011) with a separate reason code for each specific error.

13. Saved Segments in Installation Process

Prior to V3R4, you could install into saved segments during the installation process (with the I5688103 EXEC), or at post installation time.

In V3R4, this step is no longer in the I5688013 EXEC. Installing into saved segments must be done afterwards.

If you have automated the running of this EXEC by providing an input file containing the answers to the prompts (rather than submitting them from the console), the EXEC will fail when trying to process your input to the removed saved segment step. You will have to modify your answer file accordingly.

14. Enhancement to COLDLOG

Prior to V3R4, the COLDLOG reconfiguration function erased the log contents before starting the database manager. No warning was given if there were any logical units of work in the log that were needed for recovery processing.

In V3R4, the log content is not erased until after startup, and the user is warned beforehand if the log content is needed for recovery.

If you have automated the COLDLOG function in some way by providing a predetermined set of answers to the prompts (rather than submitting them from the console), the SQLLOG EXEC will fail. You will have to modify your automated process to accommodate the change. See the *DB2 Server for VM System Administration* manual for more information on this function.

Detailed Notes on V3R4-V3R3 Incompatibilities

1. Data Types of Parameter Markers in Predicates

In this first example, prior releases would resolve the data type of the parameter marker as DEC(4,2), whereas V3R4 would resolve it as INTEGER (assuming INTEGERCOL is the name of a column with a data type of INTEGER).

```
23.55 BETWEEN ? AND INTEGERCOL
```

The next two examples illustrate how these data type differences can produce quite different end results when the SQL statement is executed. In this next example, the predicate would generate SQLCODE -302 (SQLSTATE 22003) in

prior releases, when the leftmost parameter marker is assigned a value of 345 and the rightmost parameter marker is assigned a value of 206.7. This error will not occur in V3R4.

```
EDLEVEL IN (16, ?, 17.3, ?)
```

This is because the prior releases assign a data type of DEC(3,1) to the rightmost parameter marker, to which the value 206.7 cannot be assigned. V3R4 assigns a data type of SMALLINT to the rightmost parameter marker (based on the column EDLEVEL) and then truncates 206.7 to accommodate this data type.

In the next example, the predicate would generate SQLCODE -302 (SQLSTATE 22001) in V3R4, but not in prior releases, when the parameter marker is assigned a value of 'GHIJKL'.

```
DEPTNO IN ('ABCDEF', ?, 'ABC')
```

This is because V3R4 assigns a data type of CHAR(3) to the parameter marker (based on column DEPTNO), to which the value 'GHIJKL' cannot be assigned. Prior releases assign a data type of CHAR(6) to the parameter marker.

V3R5 and V3R4 Incompatibilities

1. SQL/DS Database Archive Incompatibilities

Archives that were created on prior releases of SQL/DS cannot be restored by the SQL/DS V3R5 database manager. If this is attempted, the database manager will issue message ARI2038E and terminate. See the *DB2 Server for VM Messages and Codes* or *DB2 Server for VSE Messages and Codes* manual for more details on this message.

2. SQL/DS VSAM Shareoptions Changes under VSE

In prior releases of SQL/DS (VSE), the VSAM SQL/DS directory, data and log data sets were defined with SHAREOPTIONS(1). In SQL/DS V3R5, these VSAM files must now be defined with SHAREOPTIONS(2).

3. SQLSTATE Values Changes

Many SQLSTATE values have changed in SQL/DS V3R5. The new SQLSTATE values and their former values can be found in the *DB2 Server for VM Messages and Codes* or *DB2 Server for VSE Messages and Codes* manuals. Changing SQLSTATEs is an incompatible change since many SQLSTATE values that are returned from diagnostic situations will be different from previous releases of SQL/DS. Application programmers should review any programs that use SQLSTATE in the SQLCA each time an SQL statement is executed.

4. Messages and Codes Changes

Some SQL/DS messages and codes have changed, and some new ones have been added in SQL/DS V3R5. See the *DB2 Server for VM Messages and Codes* and *DB2 Server for VSE Messages and Codes* manuals for details.

5. Display CICS Information on SHOW CONNECT

If the package that the connected user is running was created in SQL/DS Version 2 Release 2 or earlier, the CICS information will not be displayed by the SHOW CONNECT command because the RDIIN for V2R2 or earlier does

not contain the RDIIN extension area. The package must be reprepiped with SQL/DS V3R5 and recompiled to make the CICS information available.

V5R1 and V3R5 Incompatibilities

1. Messages and Codes Changes

Many messages and codes have changed, and some new ones have been added in DB2 Server for VSE & VM Version 5 Release 1. See the *DB2 Server for VM Messages and Codes* and *DB2 Server for VSE Messages and Codes* manuals.

2. DB2 Database Archive Incompatibilities

Archives that were created on prior releases cannot be restored by the DB2 Server for VSE & VM Version 5 Release 1 database manager. If this is attempted the database manager will issue message ARI2038E and terminate. See the *DB2 Server for VM Messages and Codes* manual.

3. DBSU

If you use R350 DBSU to unload and reload a table in a R510 database, the value of the DATACAPTURE column will be lost.

4. Date/Time Exits and Field Procedures

VM Users with Date/Time or Field Procedure Exits that are dependant on running in a 370 Mode virtual machine must convert to execute in a ESA mode virtual machine. Note that exits requiring AMODE=24 are not affected, as we still support running the Server code in AMODE=24. The above also applies to Single User Mode application programs. The above also applies to Vendor programs that run on the Server, such as database monitoring or tape mount handling programs.

V6R1 and V5R1 Incompatibilities

1. Running the Database Server in 24-bit Addressing Mode (VM)

With Version 6 Release 1 the RDS component is linkedited with the AMODE ANY option, instead of AMODE 24. This allows RDS to be loaded and executed above the 16MB line. This will free up valuable storage below the 16 MB line. However, if you use the AMODE(24) parameter, then RDS cannot be executed above the line. If this is attempted, a program check will occur at start up time.

To avoid this, you must use a maximum virtual storage size of 16MB which will force RDS to be loaded below the line. If you need to run with AMODE(24) all of the time, you should create an RDS saved segment that resides below the 16MB line. If you only use AMODE(24) some of the time, such as with some single user mode applications, you can create an alternate bootstrap package which specifies an alternate RDS saved segment which resides below the 16MB line, or specifies that RDS is run from free storage.

The AMODE parameter value is saved in the "resid SQLDBN Q" file. See the *DB2 Server for VM System Administration* manual for details on the AMODE parameter and saved segments.

Bibliography

This bibliography lists publications that are referenced in this manual or that may be helpful.

DB2 Server for VM Publications

- *DB2 Server for VM Application Programming*, SC09-2661
- *DB2 Server for VM Database Administration*, SC09-2654
- *DB2 Server for VSE & VM Database Services Utility*, SC09-2663
- *DB2 Server for VM Diagnosis Guide and Reference*, LC09-2672
- *DB2 Server for VSE & VM Overview*, GC09-2806
- *DB2 Server for VSE & VM Interactive SQL Guide and Reference*, SC09-2674
- *DB2 Server for VM Master Index and Glossary*, SC09-2666
- *DB2 Server for VM Messages and Codes*, GC09-2664
- *DB2 Server for VSE & VM Operation*, SC09-2668
- *DB2 Server for VSE & VM Quick Reference*, SC09-2670
- *DB2 Server for VM System Administration*, SC09-2657

DB2 Data Spaces Support Publications

- *DB2 Server Data Spaces Support for VM/ESA*, SC09-2675

Related Publications

- *DB2 Server for VSE & VM Data Restore*, SC09-2677
- *DRDA: Every Manager's Guide*, GC26-3195
- *IBM SQL Reference, Version 2, Volume 1*, SC26-8416
- *IBM SQL Reference*, SC26-8415

VM/ESA Publications

- *VM/ESA: General Information*, GC24-5745
- *VM/ESA: VMSES/E Introduction and Reference*, GC24-5837
- *VM/ESA: Installation Guide*, GC24-5836
- *VM/ESA: Service Guide*, GC24-5838
- *VM/ESA: Planning and Administration*, SC24-5750
- *VM/ESA: CMS File Pool Planning, Administration, and Operation*, SC24-5751

- *VM/ESA: REXX/EXEC Migration Tool for VM/ESA*, GC24-5752
- *VM/ESA: Conversion Guide and Notebook*, GC24-5839
- *VM/ESA: Running Guest Operating Systems*, SC24-5755
- *VM/ESA: Connectivity Planning, Administration, and Operation*, SC24-5756
- *VM/ESA: Group Control System*, SC24-5757
- *VM/ESA: System Operation*, SC24-5758
- *VM/ESA: Virtual Machine Operation*, SC24-5759
- *VM/ESA: CP Programming Services*, SC24-5760
- *VM/ESA: CMS Application Development Guide*, SC24-5761
- *VM/ESA: CMS Application Development Reference*, SC24-5762
- *VM/ESA: CMS Application Development Guide for Assembler*, SC24-5763
- *VM/ESA: CMS Application Development Reference for Assembler*, SC24-5764
- *VM/ESA: CMS Application Multitasking*, SC24-5766
- *VM/ESA: CP Command and Utility Reference*, SC24-5773
- *VM/ESA: CMS Primer*, SC24-5458
- *VM/ESA: CMS User's Guide*, SC24-5775
- *VM/ESA: CMS Command Reference*, SC24-5776
- *VM/ESA: CMS Pipelines User's Guide*, SC24-5777
- *VM/ESA: CMS Pipelines Reference*, SC24-5778
- *VM/ESA: XEDIT User's Guide*, SC24-5779
- *VM/ESA: XEDIT Command and Macro Reference*, SC24-5780
- *VM/ESA: Master Index and Glossary*, SC09-2398
- *VM/ESA: Quick Reference*, SX24-5290
- *VM/ESA: Performance*, SC24-5782
- *VM/ESA: Dump Viewing Facility*, GC24-5853
- *VM/ESA: System Messages and Codes*, GC24-5841
- *VM/ESA: Diagnosis Guide*, GC24-5854
- *VM/ESA: CP Diagnosis Reference*, SC24-5855
- *VM/ESA: CP Diagnosis Reference Summary*, SX24-5292
- *VM/ESA: CMS Diagnosis Reference*, SC24-5857

- *VM/ESA: CMS Data Areas and Control Blocks*, SC24-5858
- *VM/ESA: CP Data Areas and Control Blocks*, SC24-5856
- *IBM VM/ESA: CP Exit Customization*, SC24-5672
- *VM/ESA REXX/VM User's Guide*, SC24-5465
- *VM/ESA REXX/VM Reference*, SC24-5770

C for VM/ESA Publications

- *IBM C for VM/ESA Diagnosis Guide*, SC09-2149
- *IBM C for VM/ESA Language Reference*, SC09-2153
- *IBM C for VM/ESA Compiler and Run-Time Migration Guide*, SC09-2147
- *IBM C for VM/ESA Programming Guide*, SC09-2151
- *IBM C for VM/ESA User's Guide*, SC09-2152

Other Distributed Data Publications

- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Implementation Programmer's Guide*, SC21-9529
- *VM/Directory Maintenance Licensed Program Operation and User Guide Release 4*, SC23-0437
- *IBM Distributed Relational Database Architecture Reference*, SC26-4651
- *IBM Systems Network Architecture, Format and Protocol*
- *SNA LU 6.2 Reference: Peer Protocols*
- *Reference Manual: Architecture Logic for LU Type 6.2*
- *IBM Systems Network Architecture, Logical Unit 6.2 Reference: Peer Protocols*
- *Distributed Data Management (DDM) List of Terms*
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526
- *IBM Distributed Data Management (DDM) Architecture, Architecture Reference, Level 3*, SC21-9526

CCSID Publications

- *Character Data Representation Architecture, Executive Overview*, GC09-2207

- *Character Data Representation Architecture Reference and Registry*, SC09-2190

DB2 Server REXSQL Publications

- *DB2 REXX SQL for VM/ESA Installation*, GC09-2660
- *DB2 REXX SQL for VM/ESA Reference*, SC09-2676

C/370 Publications

- *IBM C/370 Installation and Customization Guide*, GC09-1387
- *IBM C/370 Programming Guide*, SC09-1384

Communication Server for OS/2 Publications

- *Up and Running!*, GC31-8189
- *Network Administration and Subsystem Management Guide* SC31-8181
- *Command Reference*, SC31-8183
- *Message Reference*, SC31-8185
- *Problem Determination Guide*, SC31-8186

Distributed Database Connection Services (DDCS) Publications

- *DDCS User's Guide for Common Servers*, S20H-4793
- *DDCS for OS/2 Installation and Configuration Guide* S20H-4795

VTAM Publications

- *VTAM Messages and Codes*, SC31-6493
- *VTAM Network Implementation Guide*, SC31-6494
- *VTAM Operation*, SC31-6495
- *VTAM Programming*, SC31-6496
- *VTAM Programming for LU 6.2*, SC31-6497
- *VTAM Resource Definition Reference*, SC31-6498
- *VTAM Resource Definition Samples*, SC31-6499

CSP/AD and CSP/AE Publications

- *Developing Applications*, SH20-6435
- *CSP/AD and CSP/AE Installation Planning Guide*, GH20-6764
- *Administering CSP/AD and CSP/AE on VM*, SH20-6766
- *Administering CSP/AD and CSP/AE on VSE*, SH20-6767
- *CSP/AD and CSP/AE Planning*, SH20-6770
- *Cross System Product General Information*, GH23-0500

Query Management Facility (QMF) Publications

- *QMF General Information*, GC26-4713
- *QMF VSE/ESA Setup and Usage Guide*, GG24-4196
- *Managing QMF for VSE/ESA*, SC26-3252
- *Installing QMF on VSE/ESA*, SC26-3254
- *QMF Learner's Guide*, SC26-4714
- *QMF Advanced User's Guide*, SC26-4715
- *QMF Reference*, SC26-4716
- *Installing QMF on VM*, SC26-4718
- *QMF Application Development Guide*, SC26-4722
- *QMF Messages and Codes*, SC26-4834
- *Using QMF*, SC26-8078
- *Managing QMF for VM/ESA*, SC26-8219

DL/I DOS/VS Publications

- *DL/I DOS/VS Application Programming*, SH24-5009

COBOL Publications

- *VS COBOL II Migration Guide for VSE*, GC26-3150
- *VS COBOL II Migration Guide for MVS and CMS*, GC26-3151
- *VS COBOL II General Information*, GC26-4042
- *VS COBOL II Language Reference*, GC26-4047
- *VS COBOL II Application Programming Guide*, SC26-4045
- *VS COBOL II Application Programming Debugging*, SC26-4049

- *VS COBOL II Installation and Customization for CMS* SC26-4213
- *VS COBOL II Installation and Customization for VSE* SC26-4696
- *VS COBOL II Application Programming Guide for VSE* SC26-4697

Data Facility Storage Management Subsystem/VM (DFSMS/VM) Publications

- *DFSMS/VM User's Guide*, SC26-4705

Systems Network Architecture (SNA) Publications

- *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084
- *SNA Format and Protocol Reference: Architecture Logic for LU Type 6.2*, SC30-3269
- *SNA LU 6.2 Reference: Peer Protocols*, SC31-6808
- *SNA Synch Point Services Architecture Reference* SC31-8134

Miscellaneous Publications

- *IBM 3990 Storage Control Planning, Installation, and Storage Administration Guide*, GA32-0100
- *Dictionary of Computing*, ZC20-1699
- *APL2 Programming: Using Structured Query Language*, SH21-1056
- *ESA/390 Principles of Operation*, SA22-7201

Related Feature Publications

- *Control Center Installation and Operations Guide for VM*, GC09-2678
- *IBM Replication Guide and Reference*, S95H-0999

Index

Numerics

3370 storage device

- directory size 17
- production minidisk size 8
- service minidisk size 8
- work minidisk size 8

3375 storage device

- directory size 17
- production minidisk size 8
- service minidisk size 8
- work minidisk size 8

3380 storage device 166

- directory size 17
- production minidisk size 8
- service minidisk size 8
- work minidisk size 8

3390 storage device

- directory size 17
- production minidisk size 8
- service minidisk size 8
- work minidisk size 8

370 mode 95

9332 storage device

- directory size 17
- production minidisk size 8
- service minidisk size 8
- work minidisk size 8

9335 storage device

- directory size 17
- production minidisk size 8
- service minidisk size 8
- work minidisk size 8

9345 storage device

- directory size 17
- production minidisk size 8
- service minidisk size 8
- work minidisk size 8

A

ABNEXIT

- considerations 92
- DB2 Server for VM usage 92

abnormal end

- CICS code AEY9 104, 138
- CICS code ASP7 138, 140
- CICS code ASRA 138
- exit 92
- in a field procedure 390
- log-full condition 74
- SLOGCUSH 74

ACCESS

- restriction 148

ACCOUNT initialization parameter 64

accounting facility

- account numbers for users 355
- accounting exits 355
- dbspace 271
- displaying statistics 65
- enabling 64, 65
- exit 360
- generating records 263
- introduction 261
- loading data 274
- maintaining accounting data 270
- OPTION statement 261
- records 265
- starting 262
- tables 271
- using 261

accounting record format

- CMS user 268
- initialization 266
- operator/checkpoint 267
- remote user 269
- termination 267
- VSE guest user 269

ACCT operand for accounting

- directory entries
- for accounting 261

acquiring dbspace

- for packages 161

ADD DBSPACE operation

- restarting after a failure 157
- SQLADBSP EXEC 154

adding

- dbextents
 - caution regarding 176
 - initialization parameters 172
- dbspaces
 - initialization parameters 154
 - storage pools 166

additional databases 299

address

- alternate tape drive 81

addressing type

- AMODE initialization parameter 60, 83
- TRACEBUF initialization parameter 77

advanced program-to-program

communication/virtual machine

- application program usage 102
- inter-machine communications 100

agent
 priority 70

agent structures
 description 100

allocating dbextent minidisks 166

ALT option
 FILEDEF command 81

ALTER TABLE statement 391

altering
 sorting sequence of a column 387

alternate tape drive
 virtual device address 81

alternative character sets 322

AMODE parameter
 in the SQLDBN file 304
 SQLDBGEN 301
 SQLDBINS 300
 SQLLOG 242
 SQLSTART 60

AMODE(24) parameter
 SQLSTART EXEC 83

AMODE(31) parameter
 SQLSTART EXEC 83

application program
 eye-catcher usage 93
 multiple user mode 85
 running in single user mode 86

application requester
 CCSID conversion 333
 default CCSID 341
 determining CCSID 337
 setting DBCS option 344
 setting default CHARNAME 341, 342

application requester defaults
 choosing CCSID after migration 38
 choosing CHARNAME after migration 38

application server
 CCSID conversion 333
 default 26
 default subtype
 uses 26
 name 25
 setting DBCS option 343
 setting default CCSID 338
 setting default character subtype 342
 setting default CHARNAME 338

application server defaults
 choosing CCSID after migration 35
 choosing CHARNAME after migration 35

archiving
 after ADD DBEXTENT 177
 after DELETE DBEXTENT 177
 ARCHIVE command 211
 ARCHPCT initialization parameter 74, 212
 caution regarding 176
 commands 211

archiving (continued)
 concurrently 75
 database 206, 211, 215
 default FILEDEF 85
 device address 98
 disk 219
 before starting the application server 220
 disadvantages 219
 during an archive 219
 maintaining 221
 restoring 228
 risks 219
 with shared file system (SFS) 219
 effects on nonrecoverable data 253
 facilities 206
 LARCHIVE command 216
 log archive
 disk 219, 220
 introduction 207
 process 216
 log continuity 238
 log size impact 19
 LOGMODE initialization parameter
 description 73
 of unallocated pages 207
 procedures 211
 processing 206
 restart procedures 223
 restore set and history area 246
 restoring the database 224
 shutdown 97
 SQLEND ARCHIVE command 211
 SQLEND LARCHIVE command 216
 SQLEND UARCHIVE command 215
 tape damage 229
 tape requirements 10
 user archive 215
 using database manager facilities 206
 using user facilities 206

ARCHPCT
 initialization parameter
 archiving 212
 description 74
 log archiving 216
 log activity 75

ARIBFPPB mapping macro 393

ARIRCAN MACRO 386

ARISAVES EXEC
 SQLDBA bootstrap 189
 syntax 493

ARISCCS macro 62

ARISDBG MACRO V 305

ARISDBMA EXEC 414, 490

ARISDTM MACRO 369, 380

ARISFDEF EXEC 304

ARISFPKY EXEC 39
ARISNLSC MACRO 182, 201, 351
ARISPDFC 495
ARISPDFC EXEC 291
ARISQLLD LOADLIB 7
ARISSEGC MACRO 196, 197
 SQLBOOTS EXEC 494
arithmetic operator
 in syntax diagrams xiv
ARIUEXI
 parameter list 357
ARIUEXI parameter list
 associated areas 360
ARIUSRDD 87
ARIUXDT
 IBM-supplied version 374
 parameter list 371
ARIUXIT module
 branching 356
 example user version 364
 IBM-supplied version 361
 installing 366
 service considerations 368
ARIUXTM
 IBM-supplied version 376
 parameter list 371
Assembler
 preprocessor 6, 446
 release level 4
augmented OS QSAM support 79
automatic restart resynchronization 116, 121
AVS session limit considerations 282

B

backout
 nonrecoverable data 252
BALR instruction 88
bias of the dispatcher, adjusting 70
blank
 in column with a field procedure 390
BLKSIZE option 82
blocking
 specifying 82
bootstrap module, contents 199
bootstrap package
 contents 198, 199
 copied to A-disk 199
 creating 198
 definition 182
buffer
 data pages in storage 68

C

C
 preprocessor 6, 446
 release level 4
calculations
 resource utilization 443
 storage space 449
CALL/RETURN protocols for application programs
 in single user mode 91
cancel exit 383
capacity planning 15
 planning 32
catalog
 row length 457
catalog table
 average row lengths 457
 modified by CHARNAME 63
catalog table rows 459
CEMT transaction 140
changing
 the log 240
character data 333
character set
 alternative 322
 classification and translation 482
 classification table 477
 defining your own 475
 ENGLISH 324
 example 323, 477
 FRENCH 325
 GERMAN 325
 ITALIAN 327
 PORTUGUESE 476
 selection 62
 SPANISH 329
 starting application server 330
character sets
 JAPANESE (Katakana) 328
character string
 constants 331, 332, 333
character subtype
 application server default 28
 changing default 312
 changing existing column attributes 340
 CHARSUB 26, 28
 choosing default 26
 setting application server default 342
CHARNAME
 application requester default 319
 application server default 319
 catalog tables modified 63
 change to catalog tables
 SQLDBA.ROUTINE 62
 SQLDBA.STORED QUERIES 62
 SQLDBA.SYSLANGUAGE 62
 SQLDBA.SYSTEXT2 62

CHARNAME *(continued)*

- change to catalog tables *(continued)*
 - SQLDBA.SYSUSERLIST 62
- changing 63
- changing default 312
 - considerations 320
- changing pre-Euro 321
- choosing application requester default 29
- choosing application requester default after migration 38
- choosing application server default 26
- choosing application server default after migration 35
- default CCSIDs 62
- differences between international and english code pages 339
- initialization parameter 62, 329
- INTERNATIONAL 338
- setting for application requester
 - all application requesters 341
 - an application requester 342
 - hierarchy used 341
 - using SQLGLOB EXEC 341
 - using SQLINIT EXEC 342
- setting for application server 338

checklist for database generation 30**checkpoint**

- accounting 266
- before an online log archive 217
- caused by nonrecoverable storage pools 251
- CHKINTVL initialization parameter 74
- definition 74
- log archive 217
- records on the log 205

CHKINTVL

- initialization parameter 74

choose

- in syntax diagrams xiv

choosing

- application requester default CHARNAME 29
- application server default CHARNAME 26
- application server default coded character set identifier (CCSID) 26
- default character subtype 26, 28
- default coded character set identifier (CCSID) 29

CIRA syntax 113**CIRA transaction** 117, 119**CIRB syntax** 106**CIRB transaction** 117, 119, 125, 126**CIRC syntax** 122**CIRC transaction** 125, 126**CIRD transaction** 126**CIRR syntax** 123**CIRR transaction** 109, 132**CIRT syntax** 135**CIRT transaction** 109, 132**classification table**

- character sets 482
- coded character set identifier (CCSID) 348
- identifying 62

classification, character 477**clearing the log** 240, 241**CMS**

- abnormal end routine 92
- file manipulation commands 303
- FILEDEF, ALT option 81
- files for initialization parameter 94
- files for initialization parameters 94
- HELP text files 351
- initialization parameter files 60
- OS QSAM 79
- restrictions 148

CMS communications directory

- installation process 12
- reloading 12

CMS work unit 384

- accounting exit 358
 - DB2 Server for VM tables 273
 - work unit ID 268
- storage requirements 6
- supported operating systems 1
- VM/ESA features supported 1

COBOL

- preprocessor 6, 446
- release level 4

code point 333**coded character set identifier (CCSID)**

- application requester default 29, 341
- application server default 26, 62, 338
- catalog tables modified 63
- changing CHARNAME
 - reprocess 63
- changing defaults
 - summary 320
- CHARNAME change
 - SQLDBA.ROUTINE 62
 - SQLDBA.STORED QUERIES 62
 - SQLDBA.SYSLANGUAGE 62
 - SQLDBA.SYSTEXT2 62
 - SQLDBA.SYSUSERLIST 62
- CHARNAME mapping 333
- choosing a national language 319
- choosing application requester default after migration 38
- choosing application server default after migration 35
- conversion 333
- DBCS conversion 380
- default CCSIDs 62
- defining character sets 319
- determining for an application requester 337

coded character set identifier (CCSID) *(continued)*

- identifying classification table 348
- identifying translation table 348
- loading information to CMS files 490
- MCCSIDGRAPHIC 36, 37, 338
- MCCSIDMIXED 36, 37, 338
- MCCSIDSBBCS 36, 37, 338
- migration considerations 35, 339
- mixed 336
- performance overhead 27, 319, 339
- SBCS 335
- setting for application server 338
 - differences between international and english code pages 339
- uses 26
- uses for application requester default 29

coding data by field procedures 387**coding your own exit**

- accounting 360
- requirements 372
- TRANSPROC 380

coexistence

- considerations 41
- VSE and VM 44

COLDLOG operation 224, 241

- log reconfiguration and reformatting 240
- no recovery 224
- restarting from a failure 230

coldstart 304**collection**

- national language support 283, 284
- service machines 282

COMDIR

- definition 12
- installation process 12
- namefind 12
- reloading 12

command

- ARCHIVE 211
- COMMIT WORK 263
- COUNTER 69
- DISCONN 96, 149
- FORCE 121
- FORMAT 303
- LARCHIVE 216
- RESERVE 303
- ROLLBACK WORK 263
- SHOW ACTIVE 121
- SHOW LOCK MATRIX 69
- SHOW LOG
 - scheduling archives 212
 - scheduling log archives 217
 - scheduling user archives 216
- SHOW USERS 99, 102
- SQLEND 97
- SQLEND ARCHIVE 211

command *(continued)*

- SQLEND DVERIFY 97
- SQLEND LARCHIVE 216
- SQLEND QUICK 97
- SQLEND UARCHIVE 215

COMMIT WORK 204, 263**committing changes, nonrecoverable data** 251**common operator** 57**communication protocol, specifying** 61**concurrent**

- archive 75
- specifying users 66
- users, limit 66

configuration

- concepts 277
- database manager 277
- multiple database machines and multiple databases 279
- national language support 283, 284
- one database machine and one database 278
- primary database machine 286
- secondary database machine 286
- secondary production minidisk 285
- service machine and processor 283
- VSE guest sharing 314

console, DB2 Server for VMoperator's 57**constant**

- data type, character string 332, 333

CONTINUE response 227**continuity of log archives** 238**controlling**

- access to ISQL 149
- active user number 112
- data location 165
- device and channel utilization 165

conventions

- syntax diagram notation xiii

converting

- packages 35
- service machine to database machine 53

copying

- production minidisk 291

count-key-data DASD storage

- dbextent capacity 450
- directory capacity 451
- log capacity 450

COUNTER command 69**CP**

- commands
 - DISCONN 96, 149
 - LINK in SQLFDEF 304

CREATE TABLE statement 391**creating**

- bootstrap package 198
- CMS files containing initialization parameters 94
- database archive 211

creating (*continued*)
database machine 277
log archive 216
parameter file 94
SQLFDEF file 303
user archive 215
user machines 294

CSMT transaction 140

CUREXTNT

control statement
database generation 308
estimating 21
keyword control statement 307

Customer Information Control System (CICS)

CEMT transaction 140
CIRD transaction
active transactions 138
transaction information 126
CIRT transaction 135
CSMT transaction 140
DBDCCICS 111

customizing database manager 355

CVD (column value descriptor) 393, 394

D

damaged minidisk

database 207
database and log 208
log 208

damaged minidisks

damaged
log 235

DASD

failures
damaged database 207
damaged database and log 208
damaged log 208

DASD Dump Restore (DDR)

optional software 5
utility 5

data

coding by a field procedure 387

data areas

RMXC 384

data location

controlling 165

data object 151

data pages

SYS0001 460

data recovery 151

Data System Control (DSC) 151

data type

character string
constants 332, 333
subtypes 332, 333

data type (*continued*)

constants 332, 333

database

adding 287, 297
dbextents 166
dbspaces 153
archiving 206, 217
capacity planning 15, 32
defining a log 18
deleting
dbextents 169
design 151
example 225
example storage estimate
for a production database 463
for a test database 461
for an application development database 462

extents 8

generation

additional databases 277, 299
checklist 30
control statements 307
design considerations 16
keyword control statements 307
parameters 15
planning 15
process 297
regeneration 29
restarting from a failure 230
service minidisk 294
SQLDBINS EXEC 292
summary of parameters 15
summary of process 313

machine size 5

maximum size 17

maximums 20, 470

migration 33

minidisks

replacement 232

minimum sizes 8

moving

dbextents 177

log disks 179

multiple 41, 278

user implications 279

name 25

conversion 49

operator

description 57

parameters set at database generation time 15

physical

concept 152

design 152

recovery considerations 203, 259

regeneration 29

renaming 49, 53

database (*continued*)

- restoring 217, 224, 225
- sizes 17
- starter 9, 15
- storage estimating 449

database machine 1, 277

- configuration 278, 286
- creating 277
- defining 277, 292
- description 5
- example control statements 287, 293
- in a TSAF collection 280
- in an SNA network 280
 - AVS session limit considerations 282
- initial storage requirements 444
- minidisks 7
- preparation 290, 293
- primary 284, 286
 - adding 287
- PROFILE EXEC 290, 293
- secondary 284, 286
 - adding 292
- size 5
- types 284

database manager

- environment 286
- running in disconnect 96
- startup using a CMS file 60

database manager archiving facilities

- description 206
- using to archive databases 211
- using to archive logs 216

database services utility (DBS utility)

- non-DB2 Server for VM application server 417
- storage requirements 446

date

- exit 369, 370

DATE parameter of the VSE STDOPT JCC/JCS 128**datetime**

- exits 368, 371

DB2 Server for VM database manager

- database
 - design 151
 - physical design 152

DB2 Server for VM database manager

- operator
 - VSE guest sharing 104

DB2 Server for VM database manager

- operator
 - console 57
 - description 57

DBCS (double-byte character set)

- option 331
- programming languages supported 4
- requirements 4

DBCS conversion

- CCSID to CCSID 380
- TRANSPROC exit 380

DBCS option

- setting by using SQLGLOB 344
- setting by using SQLINIT 344
- setting for all application requesters 344
- setting for an application requester 344
- setting for application server 343

dbextent 8

- adding 166
- capacities of IBM DASDs 451
- caution regarding 176
- defining minidisks 166
- definition 308
- deleting 169
- determining initial requirements 24
- estimating sizes 165
- maximum number 16
- MDISK statements for adding 166
- moving 178
- purpose 8
- sizes 15, 297
- specifying initial 308

DBMODE parameter 60**DBNAME directory**

- guest sharing 103

DBNAME parameter

- specifying 59
- SQLADBEX EXEC 171
- SQLADBSP EXEC 154
- SQLCDBEX EXEC 159, 178
- SQLDBGEN EXEC 301
- SQLDBINS EXEC 300
- SQLLOG EXEC 242
- SQLSTART EXEC 83
- starting the application server 59

DBS utility (Database Services utility)

- storage requirements 6

dbspace

- acquiring for packages 161
- ADD DBSPACE operation 154
- adding 153
- concept defined 152
- definition 309
- determining internal dbspace requirements 23
- determining requirements
 - initial 21, 22
 - system dbspaces 21
 - user dbspace 22
- index in 152
- internal 24
- maximum number 16
- nonrecoverable 251
- nonrecoverable storage pools 257
- overcommitting storage 164

dbospace *(continued)*

- private 309
- public 22, 309
- requirement 456
- restarting the ADD DBSPACE operation after a failure 157
- size for system dbspaces 22
- special
 - catalog 16
 - HELP text 16, 34
 - HELPTXT 21
 - internal 16, 310
 - ISQL 21, 465
 - package 16
 - SYS0001 21
 - SYS0002 21
- specifying initial 309
- SQLADBSP EXEC 154
- SQLADBSP file 155
- system dbospace requirements 21
- table in 152
- undercommitting storage 164
- user dbospace requirements 22

DCSSID parameter

- EXECs 183
- specifying 60
- SQLADBEX EXEC 171
- SQLADBSP EXEC 154
- SQLDBINS EXEC 301
- SQLLOG EXEC 242
- SQLSTART EXEC 83
- starting the application server 60

DDR (DASD Dump Restore)

- optional software 5
- utility 5

deadlock

- problems 69

default

- changing CCSID
 - summary 320
- changing CHARNAME
 - summary 320
- in syntax diagrams xv
- multiple user mode parameters 60
- single user mode parameters 78

default server 108**default server-name 122****default server_name 125, 126, 132****defining**

- a secondary production disk 286
- additional database machines 277
- database 11
 - log 18
 - minidisks 297
- saved segments in VM/ESA ESA Feature operating system 189

defining *(continued)*

- user machines 294

DEFSEG

- example
 - resource adapter above 16 megabytes 194
- using 189

deleting

- dbextents
 - from storage pools 169
 - initialization parameters 172

design

- database generation considerations 16

determining

- initial dbextent requirements 24
- internal dbospace requirements 23

DIRBLK parameter

- SQLDBINS 300

direct access storage device (DASD)

- directory capacity 451
- space requirements 7
- storage capacities 449
 - dbextent capacity 450
 - directory capacity 451
 - log capacity 450
- storage for the DB2 Server for VM production minidisk 8
- storage for the DB2 Server for VM service minidisk 8

directing the log archive to disk 219**directory**

- additional database machine entries 293
- allocation considerations 17
- allocations and database size 17
- capacities of IBM DASDs 451
- CMS communications 12
- delaying changes 44
- example control statements 287
 - user machine 295
- EXPAND DIRECTORY operation 159
- expansion 159
- minidisk 9
- purpose 8
- size 16, 297
 - calculating 455
 - defining 16
- space estimating 455
- SQLCDBEX EXEC 159
- verifying 97
- volume considerations 18

disciplines of dispatching 70**DISCONN 96, 149****disconnecting**

- security 149

disconnecting the database manager 96**discontiguous saved segment**

- defining components 181

discontiguous shared segment
 defining components 181

disk
 log archive 219
 log archiving to 10

DISK LOAD
 restriction 148

dispatcher 70

DISPBIAS initialization parameter 70

display terminal
 requirements 10

displaying
 transaction information 126

distributed relational database
 restrictions 499, 501

distributed relational database architecture
 installing DRDA code 413
 removing DRDA code 413

DMKSNT
 example entries for saved segments (base code) 189
 example entries for saved segments (DRDA) 189

double-byte character set (DBCS)
 identifiers 331

DRDA protocol
 benefits 411
 checklist 412
 DBS utility 417
 installing DRDA code 413
 ISQL 418
 parameter 61
 removing DRDA code 413
 responsibilities 412
 restrictions 499, 501

drive
 alternate tape 81

DSC (Data System Control) 151

DSPSTATS initialization parameter 65

dual copy 239

dual logging
 placement of logs 19
 using 239

DUALLOG initialization parameter 239

DUMPTYPE initialization parameter 76

DVERIFY parameter 97

E

END RESTORE response 227

ENGLISH character set (CCSID=37) 324

entry point of DB2 Server for VM user programs 88

environment considerations
 VM/ESA operating system 286

equipment failures 228

ERASE
 restriction 148

ESCALATE counter 69

escalation 68

estimating
 catalog dbspace (SYSTEM.SYS0001) 456
 CUREXTNT 21
 dbextent sizes 165
 dbspace size for routines 465
 dbspace sizes for stored SQL statements 466
 directory space requirement 455
 internal dbspace requirements 23
 ISQL dbspace requirements 465
 MAXDBSPC parameter 20
 MAXEXTNT parameter 20
 MAXPOOLS parameter 20
 storage for a database 449
 storage pool requirements 455
 storage pool sizes 165

EUC (Extended UNIX Code) 344

example
 adding dbextents 168
 adding dbspaces 158
 DBS utility commands
 creating accounting tables 272
 loading accounting tables 274
 defining VM minidisks for a database 297
 deleting dbextents 170
 equivalent minidisk sizes on different devices 452
 field procedure 400
 initialization parameters in CMS file 94
 starting the application server in single user mode 87
 SYS0001 storage estimating formula 461

EXEC
 ARISAVES 189
 ARISDBMA 490
 ARISFDEF 304
 ARISFPKY 39
 ARISMEX 223
 ARISPDFC 291, 495
 SQLADBEX 171, 223
 example 168, 170
 SQLADBS 154, 223
 example 158
 SQLBINS 223
 SQLBOOTS 494
 SQLCDBEX 159, 178
 SQLCIRO 223
 SQLDBGEN 223, 301
 SQLDBINS 299, 300
 SQLGENLD 199
 SQLLOG 223, 241
 SQLSTART 223
 STARTUP parameter 223

EXECIO
 restriction 148

EXECS 496

- description 497
- SQLBOOTS 496
- SQLGENLD 496
- syntax 496

execution machine

- description 5

exit

- installation 355

exit point

- for field procedures 388

exits

- abnormal end 92
- accounting 360
- cancel 383
- coding your own 372
- date 369, 370
- time 369, 370

EXPAND DIRECTORY operation

- SQLCDBEX EXEC 159

EXTEND initialization parameter 76**F****failure**

- DASD
 - damaged database 207
 - damaged database and log 208
 - damaged log 208
- equipment 228
- restarting the ADD DBSPACE operation 157
- system 223

fair-share auditing 71**fast DB2 Server for VM shutdown 99****field procedure**

- abnormal end 390
- control blocks
 - CVD 394
 - FPIB 394
 - FPPL 393
 - FPPVL 395
 - FVD 394
- description 387
- example 400
- exit point 388
- value descriptors 394

field procedure information block (FPIB) 394**field procedure parameter list (FPPL) 393****field procedure parameter value list (FPPVL) 395****field procedures**

- data type considerations 389

field-decoding

- definition 387
- input and output 399

field-definition

- definition 387

field-definition (continued)

- input and output 396

field-encoding

- definition 387
- input and output 397

FIELDPROC

- clause of CREATE or ALTER TABLE 388

FIELDPROC clause 391**file support for DB2 Server for VM database manager 82****FILEDEF**

- ALT option 81
- issued by SQLSTART 87
- issued by SQLSTART EXEC 85
- log archive to disk 85, 220
- tape 79

FILEDEF command 81**files**

- CMS
 - initialization parameters 60
- CMS commands 303
- loading DB2 Server for VM 11
- SQLADBEX 173
- SQLADBEX file 174
- SQLADBSP 155
- SQLDBN 304, 314
- SQLFDEF 172, 303, 313

FINDSYS

- diagnose instruction 88

fixed-block architecture devices 8

- dbextent capacity 451
- log capacity 451

folding rules

- coded character set identifier (CCSID) 348
- SBCS character set 319
- TRANSLATE scalar function 322

FORCE 121**FORMAT**

- data passed to a field procedure (FPPVL) 395
- FPIB 394
- restriction 148
- value descriptors (CVDs) 394

formatting

- database minidisks 303

FORTTRAN

- preprocessor 6, 446
- release level 4

FPIB (field procedure information block) 394**FPPL (field procedure parameter list) 393****FPPVL (field procedure parameter value list) 393, 395****fragment of syntax**

- in syntax diagrams xvi

FREEPCT column 162

- SYSDBSAPACES 162

FRENCH character set 325
FSERASE macro
restriction 148
FWRITE macro
restriction 148
FVD (field value descriptor) 393, 394

G

general file support 82
generating a database 15
additional database machines 277
planning 15
GERMAN character set 326
GLOBAL resource 60
guest sharing
accessing and operator responsibilities 103
DBNAME directory 103
starting 103
VM/ESA features supported 1

H

hardware requirements 6
header pages
SYS0001 461
HELP text
CMS files 351
installation prompt 292
loading
SQLDBINS EXEC 312
multiple language 320
national language 348, 353
service disk storage 8
HELPTEXT dbspace 21
hexadecimal values of the sample character sets 323
history area 244
host variable
in syntax diagrams xiv

I

identifiers
national language 350
immediate DB2 Server for VM shutdown 99
implicit CONNECT support 111
in-doubt logical units of work 116, 119, 121
incompatibilities, release to release
2.1 and 1.3.5 504, 506
2.2 and 2.1 506, 510
3.1 and 2.2 510, 520
3.2 and 3.1 521, 527
3.3 and 3.2 (VM only) 527, 534
3.4 and 3.3 (VM only) 534, 539
3.5 and 3.4 539

incompatibilities, release to release (continued)
5.1 and 3.5 540
6.1 and 5.1 540
description 503

increasing the size of the logs 240

index

invalid 34
location in dbspaces 153

initial storage 443

initialization parameters

ACCOUNT 64
AMODE 60
ARCHPCT 74
CHARNAME 62
CHKINTVL 74
CMS file 94
DBMODE 60
DBNAME 59
DCSSID 60
DISPBIAS 70
DSPSTATS 65
DUMPTYPE 76
EXTEND 76
LOGMODE 73
LTIMOUT 72
maximums 469
multiple user mode 58, 84
NCSCANS 71
NCUSERS 66
NDIRBUF 68
NLRBS 68
NLRBU 68
NPACKAGE 67
NPACKPCT 67
NPAGBUF 68
overriding 60, 93
parameter file creation 94
PROCXAB 73
PROTOCOL 61
PTIMEOUT 73
SECALVER 65
SECTYPE 65
single user mode application programs 77
SOSLEVEL 75
SQLADBEX EXEC 172
STARTUP 60
SYNCPNT 65
SYSMODE 60
TCPPOINT 66
TRACCONV 76
TRACDBSS 76
TRACDRRM 76
TRACDSC 76
TRACEBUF 77
TRACRDS 76
TRACSTG 76

initialization parameters *(continued)*

TRACWUM 76

installation

defaults

changing CCSID 320

changing CHARNAME 320

exit 355, 400

installation process

CMS communications directory 12

defining a user machine 12

generating an database 11

loading DB2 Server for VM 11

planning 1

SNA NETID file 13

installation replaceable exits 369

inter-machine communications 100

Inter-User Communication Vehicle (IUCV) 100

CONNECT 101

syntax for *IDENT 46

syntax for IUCV 48

internal dbspace

format 310

requirements 23

invalid indexes 34

ISQL (Interactive Structured Query Language)

controlling access to 149

controlling active user number 112

dbspace 21

estimating dbspace requirements 465

non-DB2 Server for VM application server 418

routines

estimating dbspace size 465

stored SQL statements

estimating dbspace size 466

ITALIAN character set 327

IUCV

*IDENT statement 288

ALLOW 288

J

JAPANESE (Katakana) character set 328

K

key

language keys and language identifiers 350

keyword

in syntax diagrams xiii

keyword control statements 307

formats 307

L

LABELDEF command 81

labeling

archive tapes 223

single-volume tapes 80

tapes 80

LANGKEY 350

language key 350

languages

choosing 319, 353

messages 348

national 348

keys and identifiers 350

messages 64

SYSLANGUAGE table 349

LARCHIVE 216

LINK commands in SQLFDEF 304

LOAD

used by database manager 88

load point of DB2 Server for VM user programs 88

loading

accounting data 274

HELP text

SQLDBINS EXEC 292

message file 351

saved segments 195

sequence 87

LOADLIB

ARISQLLD 7

FILEDEF 87

LOADMOD 87

LOADSYS 88

LOCAL resource 60

lock escalation 68

lock wait timeout 72

LOCKLMT counter 69

log

activity when ARCHPCT is reached 75

allocation considerations 19

archive

checkpoint 217

continuity 238

creating 216

device address 98

example 220

FILEDEF needed 85

introduction 207

LARCHIVE command for 216

SQLEND LARCHIVE command 216

to disk 219

capacities of IBM DASDs 451

continuity 238

damaged minidisks 235

database information 151

defining 297

description 204

dual

defining 18, 297

placing 19

log (*continued*)
 dual (*continued*)
 using 239
 failure
 switching log data 244
 full processing 74
 history area 244
 increasing the size of 240
 log-full processing 74
 maximum size by DASD type 470
 mode switching 237, 241
 moving 240
 placement of dual logs 19
 purpose 8
 reconfiguration 240
 reformatting 240, 241
 replacing 235
 damaged minidisk 208
 size 16, 19
 space 74
 switching log data 244
 switching log modes 237
 usage by DBS Utility loading 19
 volume considerations 19
logical unit of work (LUW)
 in-doubt 121
 recovery
 depending on STARTUP parameter 224
LOGMODE initialization parameter
 archiving 73, 154
 restoring 226
 switching log modes 237

M

macro
 ARIRCAN 386
 ARISCCS 62
 ARISDBG 305
 ARISDTM 369, 380
 ARISNLSC 182, 201, 351
 ARISSEGC 196, 197
 SQLBOOTS EXEC 494
maintaining
 accounting data 270
 storage pools 165
managing
 storage pools 164
manipulation commands, CMS files 303
mapping macro
 ARIBFPPB 393
MAXCONN parameter 287
 controlling overloading 66
 inter-machine communications 100
 new databases 298

MAXDBSPC control statement
 establishing database maximums 20
 estimating 20
MAXDBSPC keyword control statement
 default 308
 SQLDBGEN 307
MAXEXTNT control statement
 establishing database maximums 20
 estimating 20
MAXEXTNT keyword control statement
 default 308
 SQLDBGEN 307
maximum values
 database 20
 database size 17
 system 469
MAXPOOLS control statement
 establishing database maximums 20
 estimating 20
MAXPOOLS keyword control statement
 default 308
 SQLDBGEN 307
MCCSIDGRAPHIC 36, 37, 338
MCCSIDMIXED 36, 37, 338
MCCSIDSBSCS 36, 37, 338
MDISK control statements
 allocating dbextent minidisks 166
 directory size 16
MDISK statements
 adding dbextents 166
megabytes of data on 4-kilobyte pages 454
message
 ARI0039E 76
 ARI0041E 76
 ARI0915I 157
 choosing a national language for 348
 loading message file 351
 multiple language messages 348
 multiple national languages 320
 SET LANGUAGE command 64
methods of dispatching 70
migrating from a VSE operating system 42
migration
 CCSID considerations 35
 CHARNAME considerations 35
 considerations 33
 conversion of packages 35
 databases
 multiple 41
 defaults
 changing CCSID 320
 changing CHARNAME 320
 directory space verification 34
 handling mixed data 37
 handling SBCS data 36
 HELPTEXT dbspace 34

migration *(continued)*

- invalid indexes 34
- MCCSIDGRAPHIC 36, 37
- MCCSIDMIXED 36, 37
- MCCSIDSBCS 36, 37
- mixed primary keys 39
- planning 1, 33
- resource names 49
- setting an application requester default CCSID 38
- setting an application requester default
CHARNAME 38
- setting an application server default CCSID 35
- setting an application server default
CHARNAME 35
- VM/SP to a VM/ESA operating system 49
- VM/XA operating system to a VM/ESA operating
system 44
- VSE guest sharing 44
- VSE to a VM operating system 33, 42

minidisk

- 4 096-byte blocks 449
- 512-byte blocks 449
- damaged
 - database 207
 - database and log 208
 - log 208
- DASD storage requirements for system minidisks 8
- database 8
- defining 297
 - dbextents 166
 - example 297
- directory 8
- formatting 303
- log 241
- passwords 100, 298
- primary production 284
- production 7
- protecting 148
- replacing 232
- required for a database machine 7
- requirements for the user facility subset 9
- reserving 303
- secondary production 285
- service 7
- service machine 9
- starter database 9
- system 7
- user machine 10
 - requirements 10
- work 8

mixed data

- handling after migration 37
- MCCSIDGRAPHIC 37, 338
- MCCSIDMIXED 37, 338
- MCCSIDSBCS 37, 338
- using 331

mode switching for logs 237, 241

module

- bootstrap 182
- SQLDBBT 184
- SQLISBT 184
- SQLRMBT 184

MOVEFILE

- restriction 148

moving

- dbextents 177, 178
- log disks 179

moving a database

- from one user ID to another 50

moving the logs 240

multiple database migration 41

multiple language HELP text 320

multiple language messages 320, 348

multiple user access 279

multiple user mode

- definition 57
- initialization parameters 58
- running application programs 85
- startup using a CMS file 60

multiple virtual machine mode

- definition 57

N

name

- application server 25
- changing 49
- database 25

NAMEFIND

- restrictions 12
- storage fragmentation 12

NAMESYS macro

- example SYSNAME values 188

national language 319, 353

- character set 319
- choosing 319
- keys and identifiers 350
- messages
 - and HELP text 348
 - remote support 283
 - VSE guest sharing 351
- support 319
- SYSLANGUAGE table 349

NCSCANS initialization parameter 71

NCUSERS

- limit 66, 469

NCUSERS initialization parameter

- guidelines 66
- setting 66
- VSE guest sharing 66

NDIRBUF initialization parameter 68

network
 national language support 283
NLRBS initialization parameter 68
NLRBU initialization parameter 68
non-DB2 Server for VM application server
 DBS utility 417
 ISQL 418
nonrecoverable data 252
 backing out 252
 committing 251
 dbspaces 251
 restoring 253
 rolling back 251, 252
nonrecoverable storage pool
 data placement 254
 dbspaces 257
 querying 258
normal DB2 Server for VM shutdown 98
NPACKAGE parameter 67
NPACKPCT parameter 67
NPAGBUF initialization parameter 68
NPAGBUF parameter 68
NUCEXT QUERY
 used by database manager 88
NUCXDROP
 used by database manager 88
NUCXLOAD
 used by database manager 87, 88

O

objects 151
one-phase commit 116
online resource adapter 109, 125, 137, 140
online resource adapter data areas
 RMXC 384
online support
 implicit CONNECT support 111
 operation 104
 starting 106
 stopping 135
online support for VSE guest sharing
 considerations for stopping 99
operating
 console 57
 DB2 Server for VMoperator 57
 multiple databases 57
 online support 104
 planning 57
 VSE guest sharing 104
operating mode
 changing 95
operating modes
 modes supported
 370 mode 1, 95
 XA mode 1, 95
 XC mode 1, 95

operating modes (continued)
 XA mode 356
 XC mode 356
operating systems
 overview 1
operator response during restore
 CONTINUE 227
 END RESTORE 227
 STOP SYSTEM 227
OPTION statement for accounting 261
optional
 default parameter
 in syntax diagrams xvi
 item
 in syntax diagrams xiv
 keyword
 in syntax diagrams xvi
OS QSAM
 augmented by DB2 Server for VM database manager 79
overcommitting dbspace storage 164
override file
 example 188
overriding initialization parameters 60, 93
owning a database
 definition 278

P

package 21, 161
 conversion 35, 43
 dbspace 161
 unused 163
page
 data
 SYS0001 460
 dbspace storage 152
 header
 SYS0001 461
 megabytes and 4-kilobyte pages 454
 storage buffers 68
parameter
 ACCOUNT 64
 adding dbspaces 154
 AMODE(24)
 SQLSTART EXEC 83
 AMODE(31)
 SQLSTART EXEC 83
 CHARNAME 329
 database parameters set at database generation
 time 15
 DATE 128
 DBMODE 60
 DBNAME
 specifying in SQLSTART 60
 SQLADBEX EXEC 171
 SQLADBSP EXEC 154

parameter *(continued)*

DBNAME *(continued)*

- SQLCDBEX EXEC 159, 178
- SQLDBGEN EXEC 301
- SQLDBINS EXEC 300
- SQLLOG EXEC 242
- SQLSTART EXEC parameter 83

DCSSID

- EXEC parameter 183
- SQLADBEX EXEC 171
- SQLADBSB EXEC 154
- SQLDBINS EXEC 301
- SQLLOG EXEC 242
- SQLSTART EXEC 83

DSPSTATS 65

DVERIFY 97

LOGMODE 154

MAXCONN 287

MAXDBSPC 20

MAXEXTNT 20

MAXPOOLS 20

PARM 302

POOL 171, 302

PROCMXAB 73

PTIMEOUT 73

ranges 15

RESID

- SQLDBGEN EXEC 302
- SQLDBINS EXEC 300

SECALVER 65

SECTYPE 65

STARTER 302

starter database 15

SYNCPNT 65

TCPPOINT 66

parameter file

creation 94

parentheses

in syntax diagrams xiv

PARM parameter 302

partial row updates 253

password

minidisk 100, 298

password implications 140

performance

resource trade-offs 443

physical database 152

PL/I

preprocessor 6, 446

release level 4

planning

database generation 15

database manager 10

DB2 Server for VM database machine 1

installation 1, 15

migration 1, 33

planning *(continued)*

operation 57

POOL

keyword control statement 308

parameter 171, 302

PORTUGUESE character set 476

preparing input for SQLDBGEN 301

preprocessor

storage requirements 6, 446

primary database machine 284, 286

primary production minidisk 284

priority dispatcher 70

privilege classes

user machine 295

processor storage requirements 443

PROCMXAB initialization parameter 73

product-supplied archiving facilities

description 206

using to archive databases 211

using to archive logs 216

production minidisk 7

primary 285

secondary 286

example 285

storage requirements 8

PROFILE EXEC

additional database machine 293

database machine 290, 293

user machine 296

program products required by DB2 Server for VM

database manager 4

programming languages

double-byte character set (DBCS) support 4

release level 4

protecting minidisks 148

PROTOCOL parameter 61

pseudo-agent structure 101

PTIMEOUT initialization parameter 73

public dbspace 22, 309

punctuation mark

in syntax diagrams xiv

Q

Q-disk

storage requirements 8

QSAM, as augmented by DB2 Server for VM

database manager 79

querying

nonrecoverable storage pools 258

quick DB2 Server for VM shutdown 99

R

READCARD

restriction 148

READY/RECOVERY agent 117

real agents 101

real storage requirements
summary 6

RECEIVE
restriction 148

reconfiguring the log 240

records
spanned 81

recoverable
DBS utility command 253
SQL statements 253

recovery 252
abnormal end in a field procedure 390
considerations 203, 259
DASD failures
damaged database 207
damaged database and log 208
damaged log 208
dual copy 239
dual logging 239
dual logs 224
history area 244
log reconfiguration 240
log reformatting 241
nonrecoverable data 251
nonrecoverable dbspaces 251
nonrecoverable storage pools 250, 254
processing failures with nonrecoverable data 252
special topics 237
SQLEND 99
switching log data between logs 244
switching log modes 237
system failure 223

recovery list 117

reference information for storage space calculations 449

reformatting
log 240, 241

regenerating a database 29

regular DB2 Server for VM shutdown 98

release level
coexistence 41
products required by DB2 Server for VM database manager 4

release to release incompatibilities
2.1 and 1.3.5 504, 506
2.2 and 2.1 506, 510
3.1 and 2.2 510, 520
3.2 and 3.1 521, 527
3.3 and 3.2 (VM only) 527, 534
3.4 and 3.3 (VM only) 534, 539
3.5 and 3.4 539
5.1 and 3.5 540
6.1 and 5.1 540
description 503

reloading
CMS communications directory 12
COMDIR 12

remote spooling communications subsystem (RSCS)
release required 4

Remote unit of work 1
application server
renaming 53

renaming a database 49

renaming a resource identifier 54

repeat symbol
in syntax diagrams xv

replacing
database and log minidisk 208
database minidisk 232
log minidisk 208
log minidisks 235

repository
storage requirements 6

required item
in syntax diagrams xiv

requirements
DB2 Server for VM database machine software 4

RESERVE 303
restriction for 148

reserved words
SQL xvii

resetting
the log 240, 241

RESID 25
renaming 53, 54

RESID NAMES file 26, 46, 53, 54

RESID parameter
SQLDBGGEN EXEC 302
SQLDBINS EXEC 300

resource adapter
cancel support 385
definition 356

resource adapter data areas
RMXC 384

resource and performance trade-offs 443

resource names migration 49

resource utilization calculations 443

restart recovery
definition 205

restarting
ARISMEX 223
from COLDLOG operation failure 230
from database generation failure 230
from database restore operation 228
from system failure while archiving 230
procedures 223
SQLADBEX 223
SQLADBSP 223

restarting (*continued*)
 SQLCIREO 223
 SQLSTART 223
 STARTUP parameter 223

restore set 246

restoring
 database
 restarting from failure of 228
 restore set and history area 246
 starting the application server when
 restoring 225
 STARTUP=R 225
 nonrecoverable data 253

restoring a database
 procedures 224

restriction
 CMS 148
 commands
 ACCESS 148
 DISK LOAD 148
 ERASE 148
 EXECIO 148
 FORMAT 148
 MOVEFILE 148
 READCARD 148
 RECEIVE 148
 RESERVE 148
 macros
 FSERASE 148
 FSWRITE 148

restrictions
 DRDA protocol 499, 501

RESYNCH command 121

resynchronization 126

resynchronization transaction 117

REXX 4

RMXC 384

ROLLBACK WORK 263

ROLLBACK WORK command 204

rolling back
 nonrecoverable data 252

round-robin dispatcher 70

routine
 estimating dbspace size 465

running
 application programs
 single user mode 86
 multiple user mode 85
 SQLLOG EXEC 241

RXSQL 4

S

sample
 character sets 323

saved segment
 defining 12

saved segments
 defining 187
 DMKSNT entries (base code) 189
 DMKSNT entries (DRDA) 189
 loading 195
 overlaps 188
 SEGMENT RESERVE command 181, 187, 200
 stop using 200

saved segments (DCSS)
 national language 351

saving
 segments 195

SBCS character set
 folding rules 319

SBCS data
 handling after migration 36
 MCCSIDGRAPHIC 36
 MCCSIDMIXED 36
 MCCSIDSBBCS 36

scan control blocks 71

scan table 71

SECALVER initialization parameter 65

secondary database machine 284

secondary production minidisk 285
 defining 286
 example 285

SECTYPE initialization parameter 65

security
 conversation level 144
 distributed processing
 administration 147
 security 147
 maintaining 143, 150
 session level 144
 user ID translation 147
 VM directory control statement
 IUCV *IDENT 146
 IUCV ALLOW 146
 IUCV ANY 146
 VM/ESA operating systems 146

SEGMENT FIND
 used by database manager 89

SEGMENT LOAD
 used by database manager 89

SEGMENT RESERVE command 181, 187, 200

segment saving 195

selecting
 character sets 62

sequence of loading 87

server_names
 on the CIRA transaction 114
 on the CIRB transaction 108

service disk storage requirements 8

service machine
 converting to a database machine 53
 defining 11
 example 283
 in a collection 282
 minidisks 9
 size 5

service minidisk 7

session limit (AVS) 282

SET APPCVM 103

SET MACHINE command 95

setting
 default character subtype 342

shadow paging 456

shared file system directory (SFS directory)
 production minidisk size 8
 service minidisk size 8
 work minidisk size 8

shared segment 181

SHOW ACTIVE 121

SHOW LOCK MATRIX command 69

SHOW LOG
 scheduling archives 212
 scheduling log archives 217
 scheduling user archives 216

SHOW USERS 99, 102

shutdown of the application server 97

signing off the DB2 Server for VM system console 96, 149

single console image facility (SCIF) 57

single user mode
 AMODE(24) 86, 87
 CALL/RETURN protocols for application programs 91
 definition 58
 initialization parameters 77
 running application programs 86
 starting the application server 86
 starting the application server and providing user parameters 90

single virtual machine mode
 definition 58

single-volume tapes 80

size
 database 17
 directory 16, 297
 internal dbspaces needed 23
 log 16, 19
 by DASD type 470

SLOGCUSH initialization parameter
 description 74
 freeing log space 212, 216

slots
 use of 164

SNA (see Systems Network Architecture) 280

SNA NETID file 13

software requirements 4

sorting sequence, altering by a field procedure 387

SOSLEVEL initialization parameter 75

space
 DB2 Server for VM production minidisk 8
 DB2 Server for VM service minidisk 8

space allocations
 log and dbextent minidisks 452

SPANISH character set 329

spanned records 81

specifying
 initial dbspace 309
 user parameters 90

SPIE exit
 considerations 92

SQLADBEX EXEC 171
 example 168, 170

SQLADBEX file 173, 174

SQLADBSP EXEC 22, 153, 154
 example 158
 SQLADBSP file 155

SQLADBSP file 155

SQLBOOTS EXEC 200

SQLCDBEX EXEC 159, 178

SQLCODE
 -522 72
 -937 140
 -945 163
 -946 163

SQLDBA
 bootstrap package 183
 password 313
 SQLDBBT module 184
 SQLISBT module 184
 SQLRMBT module 184
 user ID 313

SQLDBA SQLDBBT Q 496, 497

SQLDBA SQLISBT Q 496, 497

SQLDBA SQLRMBT Q 496, 497

SQLDBA.ROUTINE
 CHARNAME change 62

SQLDBA.STORED QUERIES
 CHARNAME change 62

SQLDBA.SYSLANGUAGE
 CHARNAME change 62

SQLDBA.SYSLANGUAGE table 349

SQLDBA.SYSTEXT2
 CHARNAME change 62

SQLDBA.SYSUSERLIST
 CHARNAME change 62

SQLDBGEN example 305

SQLDBGEN EXEC 301
 files used by 305
 format 301
 preparing input for 301

SQLDBGEN file 53, 314
renaming a resource identifier 55

SQLDBINS EXEC 299, 300
format 299
running 299

SQLDBN file
contents 304, 314
purpose 314
renaming a database 53
renaming a resource identifier 55
use of 184

SQLDS protocol 61

SQLEND
ARCHIVE 211
DVERIFY 97
LARCHIVE 216
QUICK 97
UARCHIVE 215

SQLEND operator command 97

SQLERRD2
-30 72

SQLFDEF file 313
creating 303
renaming a database 54
renaming a resource identifier 55
single logging 242
updated by SQLADBEX 172

SQLGENLD EXEC 182, 199

SQLGLOB EXEC
setting DBCS option 344
setting default CHARNAME 341

SQLINIT EXEC
issuing for a user 296
setting DBCS option 344
setting default CHARNAME 342

SQLLOG EXEC 229, 241

SQLSTART EXEC 83, 157
restoring the database example 225

STAE exit, considerations for 92

starter database
control statements 306

STARTER parameter 302

starting
accounting facility 262
online support 106

starting application server
after restoring the database 226
FRENCH character set example 330
restoring 225
STARTUP=R 225

starting the application server 57, 94
multiple user mode 83
single user mode 86

STARTUP initialization parameter
restart procedures 223
restoring a database 225

STDOPT JCC/JCS 128

STOP SYSTEM response 227

stopping the application server
online support 99, 135
recovery considerations 98

storage
capacity IBM DASD 449
concepts 151
devices 17
estimating 449
initial 443
initial storage formula 443
initial storage requirements 444
processor requirements 443
requirements
DB2 Server for VM production minidisk 8
DB2 Server for VM service minidisk 8
DBS utility 446
processor 443
virtual
recommended 6
virtual storage requirements
user machines 445

storage pool
adding dbextents 166
concepts 151
controlling
channel utilization 165
device utilization 165
controlling data location 165
definition 308
deleting dbextents 169
design considerations 164
estimating size 165
maintaining 165
management 164
monitoring 165
nonrecoverable 250
requirements 455
space 73
specifying initial 308
support
application dbspaces 24
internal dbspaces 25
system dbspaces 24

storage pools
definition 309

storage requirements
DBS utility 6

storage requirements (virtual and real) 443

storage space calculations 449

stored SQL statements
estimating dbspace size 466

subtype
application server uses 26

switching

- log data between logs 244
- log modes 237

SYNCPNT initialization parameter 65

SYNCPNT parameter

- multiple-site read multiple-site update 413
- multiple-site read single-site update 413

syntax diagram

- notation conventions xiii

SYS0001 dbspace 21

- storage estimating 456
 - derivation of the formula 460
 - examples 461
 - modifying the formula 463

SYS0002 dbspace 21

- allocation 161

SYS000n dbspace 163

SYSCCSIDS catalog table 323

SYSCHARSETS catalog table 323

SYSIN, valid assignments for 82

SYSLANGUAGE catalog table

- example 349

SYSMODE initialization parameter 60

SYSNAME values for NAMESYS

- example 188

SYSOPTIONS catalog table

- updating 379

SYSPRINT, valid assignments for 82

SYSPUNCH, valid assignments for 82

SYSSTRINGS catalog table 323

system

- code
 - description 182
 - dbspaces
 - size 22
 - failure 223
 - maximums 469
 - minidisk 7
 - DASD storage requirements 8
 - minidisks 7
 - signing off console 96, 149
 - software requirements 4
 - virtual storage requirements 5
- ### Systems Network Architecture (SNA)
- AVS session limit considerations 282
 - database machines 280
 - definition 280
 - NETID file 13

T

table

- character classification 477
- location in dbspaces 153

tape

- archiving 10

tape (continued)

- damage 229
- DBS utility processing 10
- FILEDEFS 79
- LABELDEFS 79
- labeled 80
- labeling 223
- requirements 10
- single-volume 80
- support 79
- tracing 10
- unlabeled 80

tape drive

- alternate virtual device address 81

TCPDPT initialization parameter 66

terminal

- requirements 10

terminal line-edit 291

- symbols 293

terminating

- online applications 140

the application server

- starting 57

time

- exit 369, 370

TRACCONV initialization parameter 76

TRACCONV parameter

- SQLSTART 76

TRACDBSS initialization parameter 76

TRACDRRM initialization parameter 76

TRACDRRM parameter

- SQLSTART 76

TRACDSC initialization parameter 76

TRACEBUF initialization parameter 77

TRACEBUF parameter

- SQLSTART 77

tracing

- default FILEDEF 85
- tape requirements 10

TRACRDS initialization parameter 76

TRACSTG initialization parameter 76

TRACSTG parameter

- SQLSTART 76

TRACWUM initialization parameter 76

TRACWUM parameter

- SQLSTART 76

transaction information

- displaying 126

transactions

- CEMT 140
- CIRD 105
- CIRT 135
- CSMT 140

TRANSLATE function

- folding rules 322

translation table
 character sets 482
 coded character set identifier (CCSID) 348
 identifying 62
Transparent Services Access Facility (TSAF) 280
 collection
 database machines 280
 definition 280
TRANSPROC exit
 coding your own 380
two-phase commit 116

U

unallocated pages
 archiving 207
undercommitting dbspace storage 164
unlabeled tapes 80
unused package 163
updating
 partial rows 253
 VM directory entries 287, 292
user archive
 creating 215
user facility subset
 converting to a database 53
 description 1
 minidisk requirements 9
user free storage 181
user machine
 additional 294
 defining 294
 example control statements 295
 minidisk requirements 10
 minidisks 10
 PROFILE EXEC 296
 size 6
 virtual storage requirements 446
user parameters
 in single user mode 90
user restore
 definition 226
using 239
 dual logging 239
using AMODE(24)
 incompatibilities 540

V

V-disk
 storage requirements 8
value
 descriptors in field procedures 394
verifying
 directory 97

virtual device address
 alternate tape drive 81
virtual disk support
 internal dbspaces 24
virtual machine
 overview 1
 size
 DB2 Server for VM database manager 5
 user 294
Virtual Machine (VM)
 accounting 261
 directory entries
 for a user machine 295
 for database machines 292
 updating 287, 292
 single console image facility (SCIF) 57
virtual storage
 initial requirements
 database machines 444
 recommended 6
 requirements
 DB2 Server for VM components 443, 446
 user machines 446
 service machine 5
 user machine 6
VM/ESA (Virtual Machine/Enterprise Systems Architecture)
 considerations 286
 DB2 Server for VM features 1
 directory entries for accounting 261
 migration from a VM/SP operating system 49
 migration from a VM/XA operating system 44
 running the database manager 95
VM/SP (Virtual Machine/System Product)
 migration to VM/ESA operating system 33
VM/XA SP (Virtual Machine/Extended Architecture System Product)
 migration from VM/SP operating system 33
 migration from VM/XA operating system 33
 migration from VSE 42
 running the database manager 95
VMBACKUP 206, 215
 optional software 5
volume considerations
 directory 18
 log 19
VSE guest sharing 314
 accessing and operator responsibilities 103
 configuration 314
 DBNAME directory 103
 migration 44
 national language messages 351
 online support 99
 release level of VSE 4
 sample configurations 315
 single processor 315

VSE guest sharing *(continued)*

TSAF 316

VM/ESA features supported 1

VTAM product 317

VSE guest user accounting record 269

VSE operating system

migration to 42

VSE STDOPT JCC/JCS 128

W

warm start of the application server 60

work units

accounting exit

CMS 273

work unit ID 268

CMS 384

accounting exit 358

supported operating systems 1

VM/ESA features supported 1

Communicating Your Comments to IBM

DB2® Server for VM
System Administration
Version 6 Release 1
Publication No. SC09-2657-00

If there is something you like—or dislike—about this book, please let us know. You can use one of the methods listed below to send your comments to IBM. If you want a reply, include your name, address, and telephone number. If you are communicating electronically, include the book title, publication number, page number, or topic you are commenting on.

The comments you send should only pertain to the information in this book and its presentation. To request additional publications or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give it to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
 - United States and Canada: 416-448-6161
 - Other countries: (+1)-416-448-6161
- If you prefer to send comments electronically, use the network ID listed below. Be sure to include your entire network address if you wish a reply.
 - Internet: torrcf@ca.ibm.com
 - IBMLink: [toribm\(torrcf\)](mailto:toribm(torrcf)@ca.ibm.com)
 - IBM/PROFS: [torolab4\(torrcf\)](mailto:torolab4(torrcf)@ca.ibm.com)
 - IBMMAIL: [ibmmail\(caibmwt9\)](mailto:ibmmail(caibmwt9)@ca.ibm.com)

Readers' Comments — We'd Like to Hear from You

**DB2® Server for VM
System Administration
Version 6 Release 1**

Publication No. SC09-2657-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? Yes No

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

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 EGLINTON AVENUE EAST
NORTH YORK ONTARIO CANADA M3C 1H7

Fold and Tape

Please do not staple

Fold and Tape



File Number: S370/4300-50
Program Number: 5648-A70



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-2657-00



Spine information:



DB2® Server for VM

System Administration

Version 6 Release 1